

# Programming and Proving with Higher Inductive Types

Dan Licata

Wesleyan University  
Department of Mathematics and Computer Science

# Constructive Type Theory

[Martin-Löf]

Three senses of constructivity:

# Constructive Type Theory

[Martin-Löf]

Three senses of constructivity:

- ✱ Non-affirmation of certain classical principles provides **axiomatic freedom**

# Synthetic geometry

## Euclid's postulates

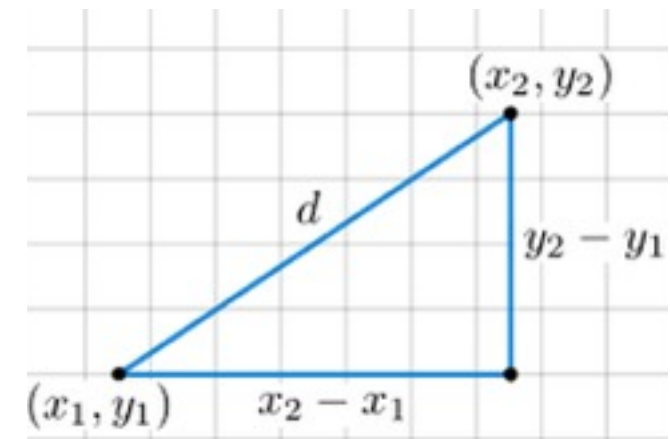
1. **To draw a straight line from any point to any point.**
2. **To produce a finite straight line continuously in a straight line.**
3. **To describe a circle with any center and distance.**
4. **That all right angles are equal to one another.**
5. **Given a line and a point not on it, there is exactly one line through the point that does not intersect the line**

# Synthetic geometry

## Euclid's postulates

1. To draw a straight line from any point to any point.
2. To produce a finite straight line continuously in a straight line.
3. To describe a circle with any center and distance.
4. That all right angles are equal to one another.
5. Given a line and a point not on it, there is exactly one line through the point that does not intersect the line

## Cartesian



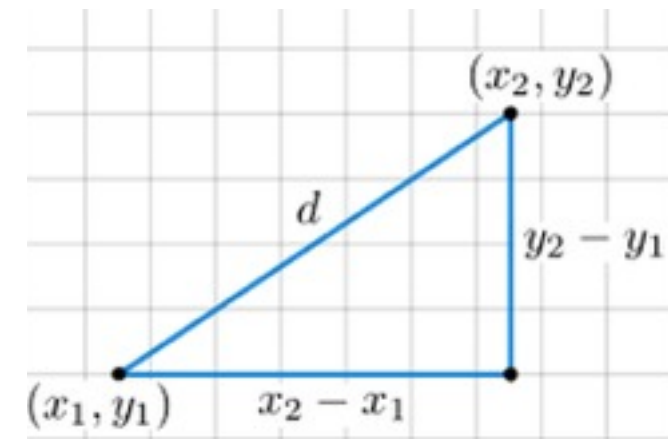
# Synthetic geometry

## Euclid's postulates

1. To draw a straight line from any point to any point.
2. To produce a finite straight line continuously in a straight line.
3. To describe a circle with any center and distance.
4. That all right angles are equal to one another.
5. Given a line and a point not on it, there is exactly one line through the point that does not intersect the line

models  
←

## Cartesian



# Synthetic geometry

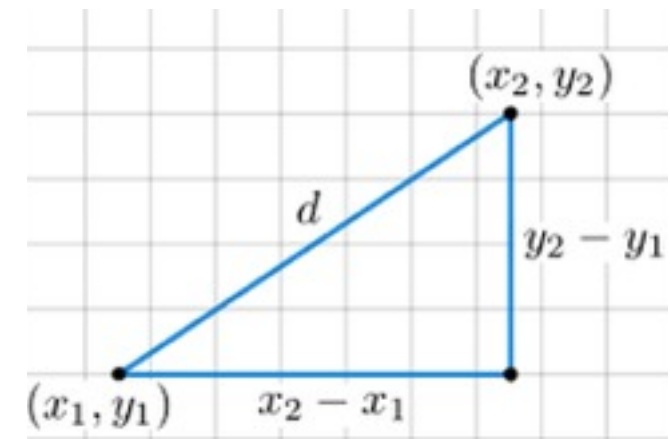
## Euclid's postulates

1. To draw a straight line from any point to any point.
2. To produce a finite straight line continuously in a straight line.
3. To describe a circle with any center and distance.
4. That all right angles are equal to one another.

models



## Cartesian



# Synthetic geometry

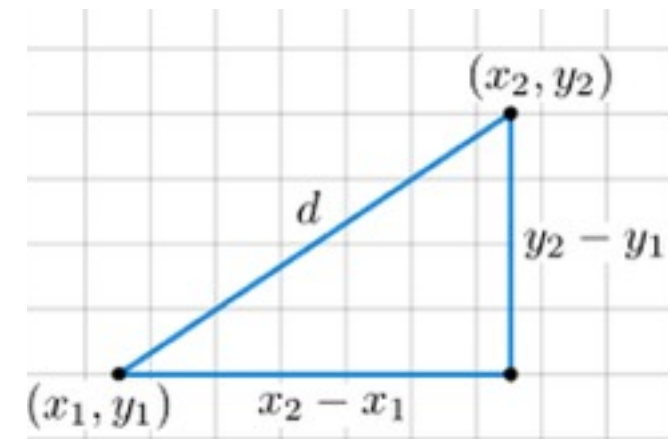
## Euclid's postulates

1. To draw a straight line from any point to any point.
2. To produce a finite straight line continuously in a straight line.
3. To describe a circle with any center and distance.
4. That all right angles are equal to one another.

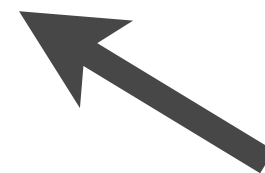
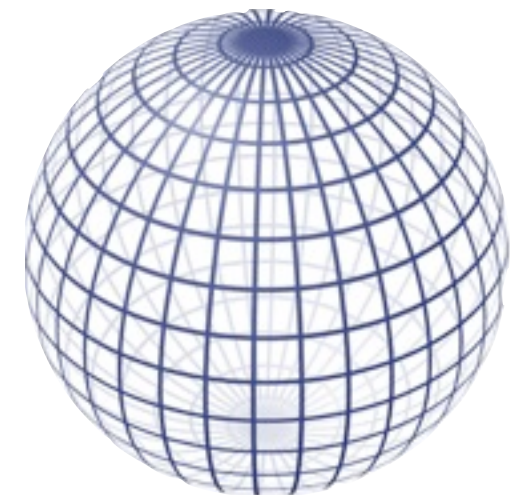
models



## Cartesian



## Spherical





# Synthetic geometry

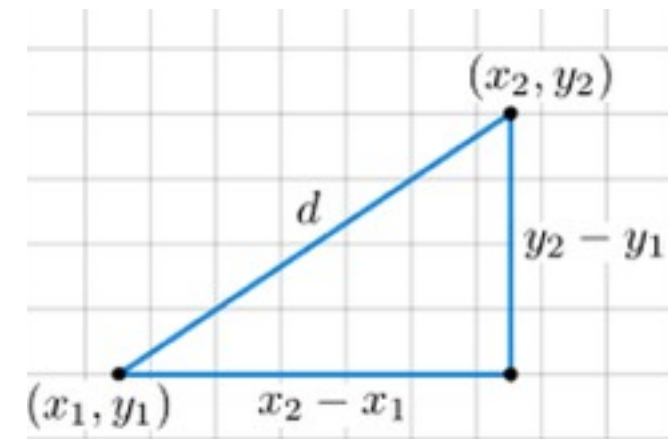
## Euclid's postulates

1. To draw a straight line from any point to any point.
2. To produce a finite straight line continuously in a straight line.
3. To describe a circle with any center and distance.
4. That all right angles are equal to one another.
5. Two distinct lines meet at two antipodal points.

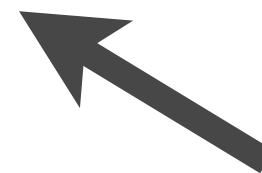
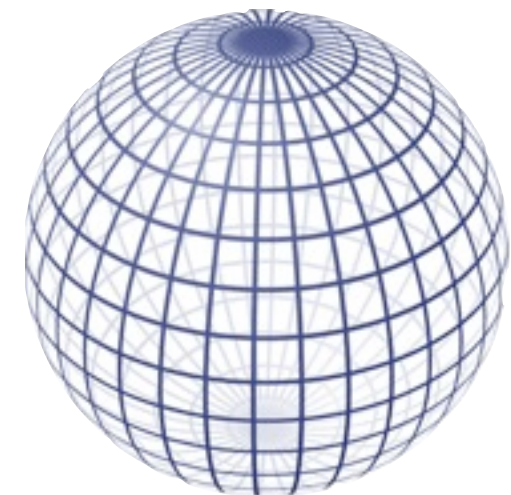
models



## Cartesian



## Spherical



# Synthetic mathematics

## Type theory

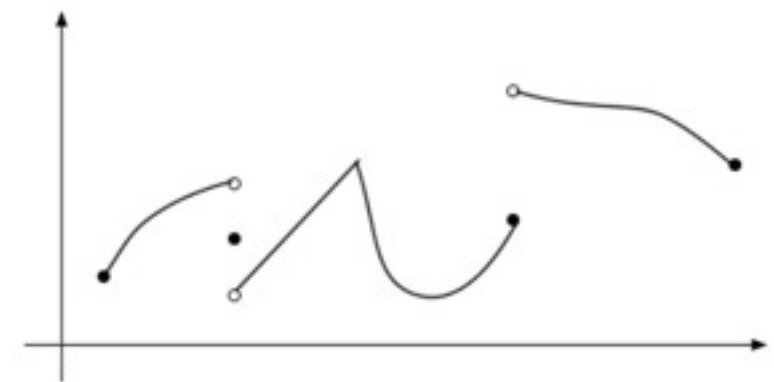
**1.  $\tau ::= \mathbf{b} \mid \tau_1 \rightarrow \tau_2$**

**2.  $e ::= \mathbf{x} \mid e_1 e_2 \mid \lambda \mathbf{x}. e$**

**3.  $(\lambda \mathbf{x}. e) e_2 = e[e_2 / \mathbf{x}]$**

# Synthetic mathematics

**Set-theoretic  
functions**



**Type theory**

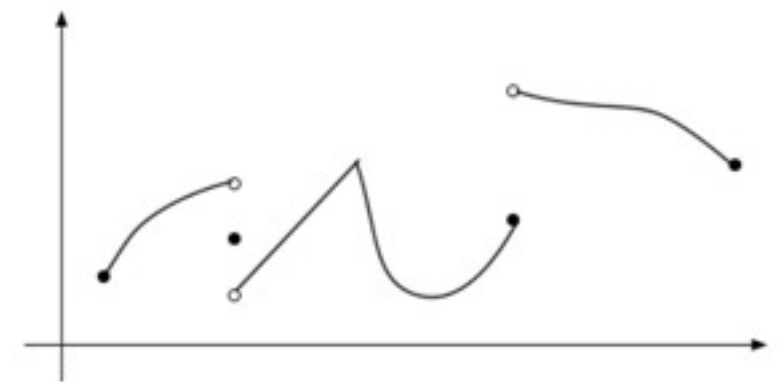
- 1.  $\tau ::= \mathbf{b} \mid \tau_1 \rightarrow \tau_2$**
- 2.  $e ::= \mathbf{x} \mid e_1 \ e_2 \mid \lambda \mathbf{x}. e$**
- 3.  $(\lambda \mathbf{x}. e) e_2 = e[e_2 / \mathbf{x}]$**

# Synthetic mathematics

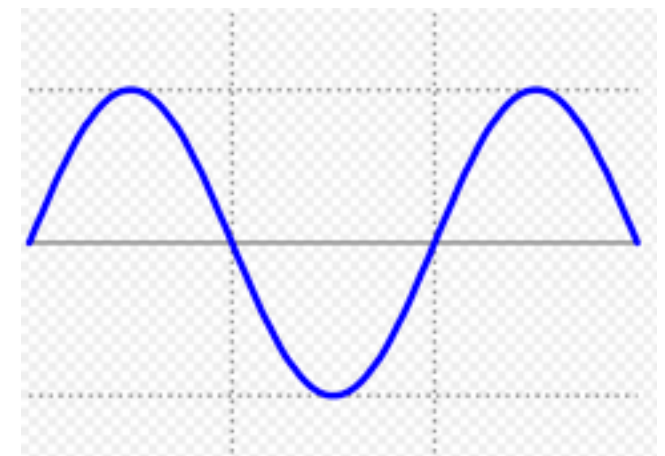
## Type theory

1.  $\tau ::= \mathbf{b} \mid \tau_1 \rightarrow \tau_2$
2.  $e ::= \mathbf{x} \mid e_1 e_2 \mid \lambda \mathbf{x}. e$
3.  $(\lambda \mathbf{x}. e) e_2 = e[e_2 / \mathbf{x}]$

## Set-theoretic functions



## Domain-theoretic functions

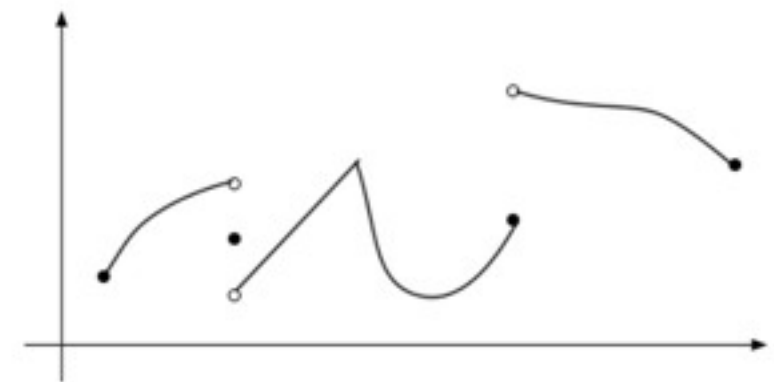


# Synthetic mathematics

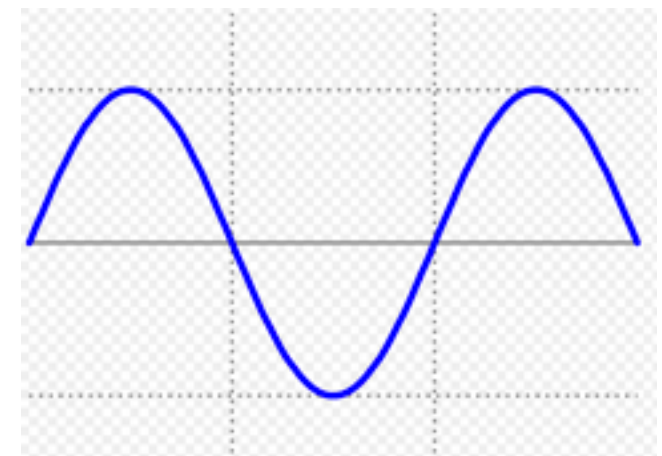
## Type theory

1.  $\tau ::= \mathbf{b} \mid \tau_1 \rightarrow \tau_2$
2.  $e ::= \mathbf{x} \mid e_1 e_2 \mid \lambda \mathbf{x}. e$
3.  $(\lambda \mathbf{x}. e) e_2 = e[e_2 / \mathbf{x}]$
4.  $\mathbf{Y}(\mathbf{f}) = \mathbf{f}(\mathbf{Y}(\mathbf{f}))$

## Set-theoretic functions



## Domain-theoretic functions



# Constructive Type Theory

Three senses of constructivity:

# Constructive Type Theory

Three senses of constructivity:

- ✱ Non-affirmation of certain classical principles provides **axiomatic freedom**

# Constructive Type Theory

Three senses of constructivity:

- ✱ Non-affirmation of certain classical principles provides **axiomatic freedom**
- ✱ **Computational interpretation** supports software verification and proof automation



# Computational Interpretation

There is an algorithm that,  
given a closed term  $e : \text{bool}$ ,  
computes either  
an equality  $e = \text{true}$ , or  
an equality  $e = \text{false}$ .

# Computational Interpretation

There is an algorithm that,  
given a closed term  $e : \text{bool}$ ,  
computes either  
an equality  $e = \text{true}$ , or  
an equality  $e = \text{false}$ .

- \* Requires functions with arbitrary domain/  
range to be computable, but stating  
theorem for  $\text{bool}$  offers some flexibility

# Computational Interpretation

There is an algorithm that,  
given a closed term  $e : \text{bool}$ ,  
computes either  
an equality  $e = \text{true}$ , or  
an equality  $e = \text{false}$ .

- \* Requires functions with arbitrary domain/  
range to be computable, but stating  
theorem for  $\text{bool}$  offers some flexibility
- \* Basis for software verification and  
proof automation

# Constructive Type Theory

Three senses of constructivity:

- ✱ Non-affirmation of certain classical principles provides **axiomatic freedom**
- ✱ **Computational interpretation** supports software verification and proof automation

# Constructive Type Theory

Three senses of constructivity:

- ✱ Non-affirmation of certain classical principles provides **axiomatic freedom**
- ✱ **Computational interpretation** supports software verification and proof automation
- ✱ Props-as-types allows **proof-relevant mathematics**

# Proof relevance

$x : A$

# Proof relevance

$x : A$

$x =_A y$

equality type

# Proof relevance

$x : A$

$p : x =_A y$       equality type



# Proof relevance

$$x : A$$
$$p : x =_A y$$

equality type

Any structure or property  $C$  can be transported along an equality

# Proof relevance

$$x : A$$
$$p : x =_A y$$

equality type

Any structure or property  $C$  can be transported along an equality

$$\text{transport}_C(p) : C(x) \rightarrow C(y)$$

# Proof relevance

$$x : A$$
$$p : x =_A y$$

equality type

Any structure or property  $C$  can be transported along an equality

$$\text{transport}_c(p) : C(x) \rightarrow C(y)$$

**Leibniz's  
indiscernability  
of identicals**



# Proof relevance

$$x : A$$
$$p : x =_A y$$

equality type

Any structure or property  $C$  can be transported along an equality

$$\text{transport}_C(p) : C(x) \rightarrow C(y)$$

**Leibniz's  
indiscernability  
of identicals**



*by a function*: can it do real work?

# Proof relevance

$x : A$

$p : x =_A y$       equality type

# Proof relevance

$x : A$

$p : x =_A y$       equality type

$p_1 =_{x=y} p_2$

# Proof relevance

$x : A$

$p : x =_A y$       equality type

$q : p_1 =_{x=y} p_2$

# Proof relevance

$x : A$

$p : x =_A y$       equality type

$q : p_1 =_{x=y} p_2$

$q_1 =_{p_1=p_2} q_2$



# Proof relevance

$x : A$

$p : x =_A y$       equality type

$q : p_1 =_{x=y} p_2$

$r : q_1 =_{p_1=p_2} q_2$

# Proof relevance

$x : A$

$p : x =_A y$       equality type

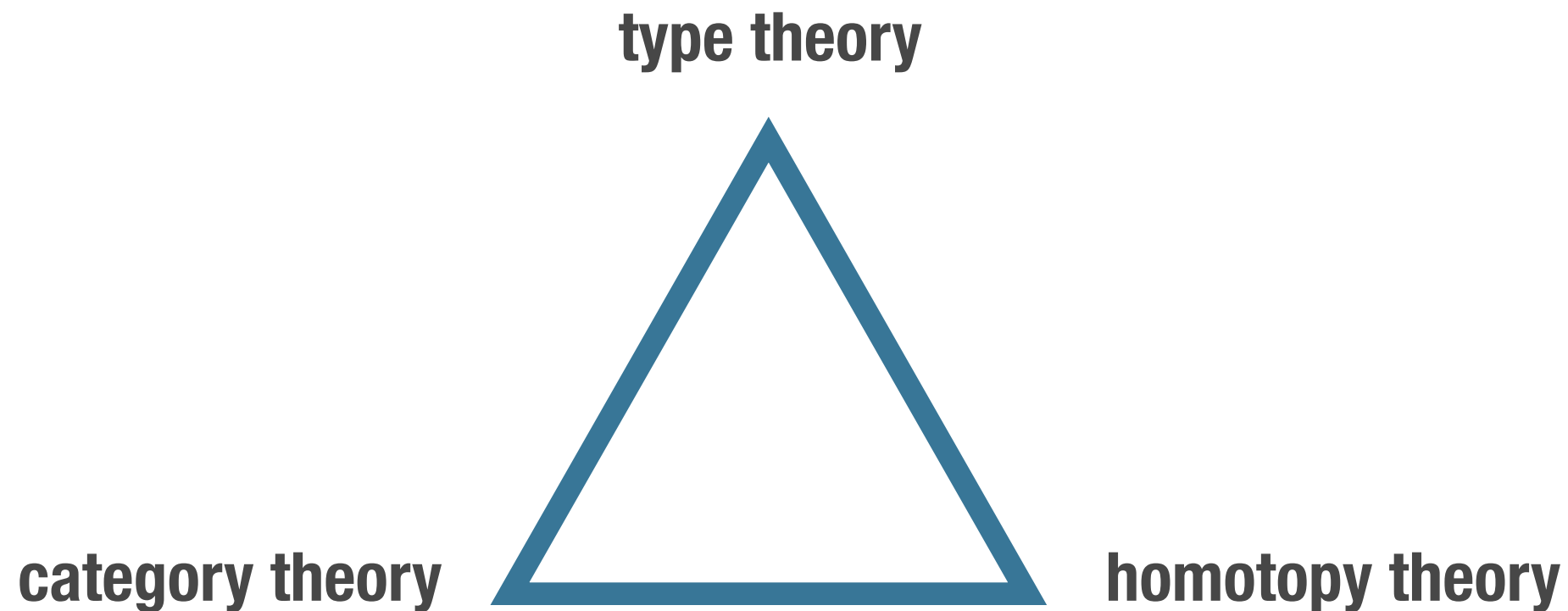
$q : p_1 =_{x=y} p_2$

$r : q_1 =_{p_1=p_2} q_2$

$\vdots$

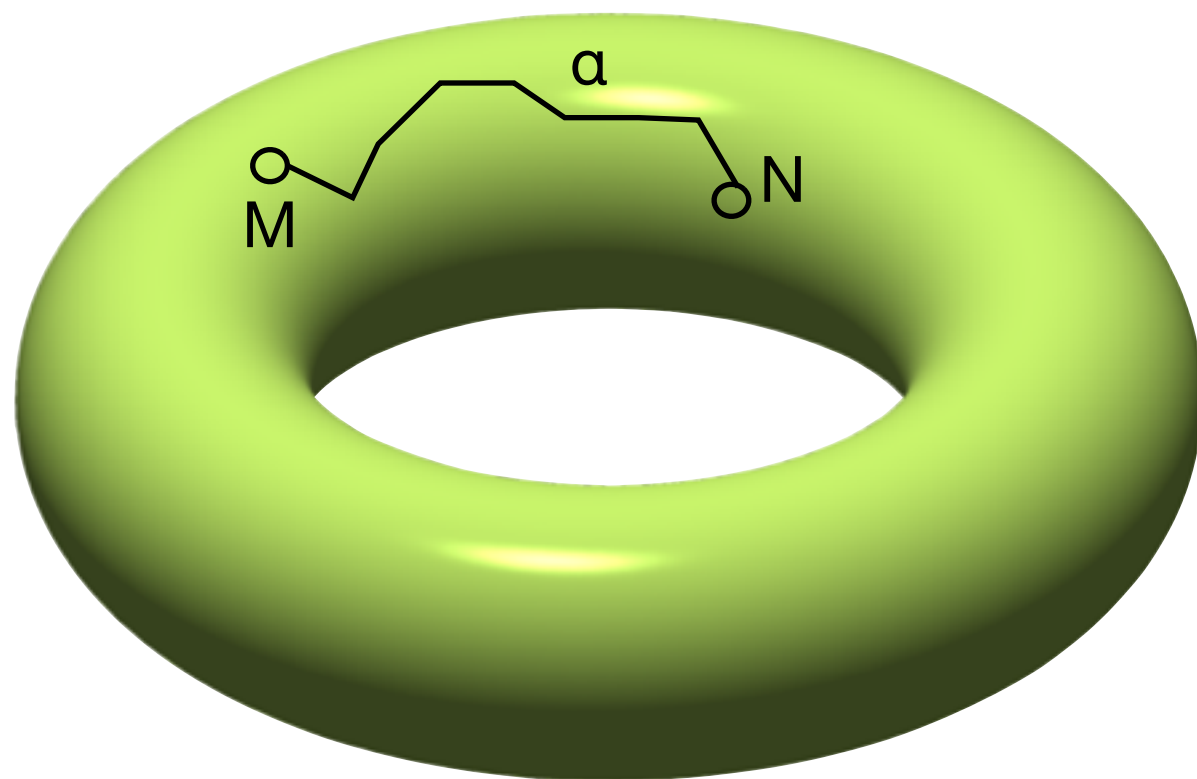
*higher equalities radically expand the kind of math that can be done synthetically...*

# Homotopy Type Theory



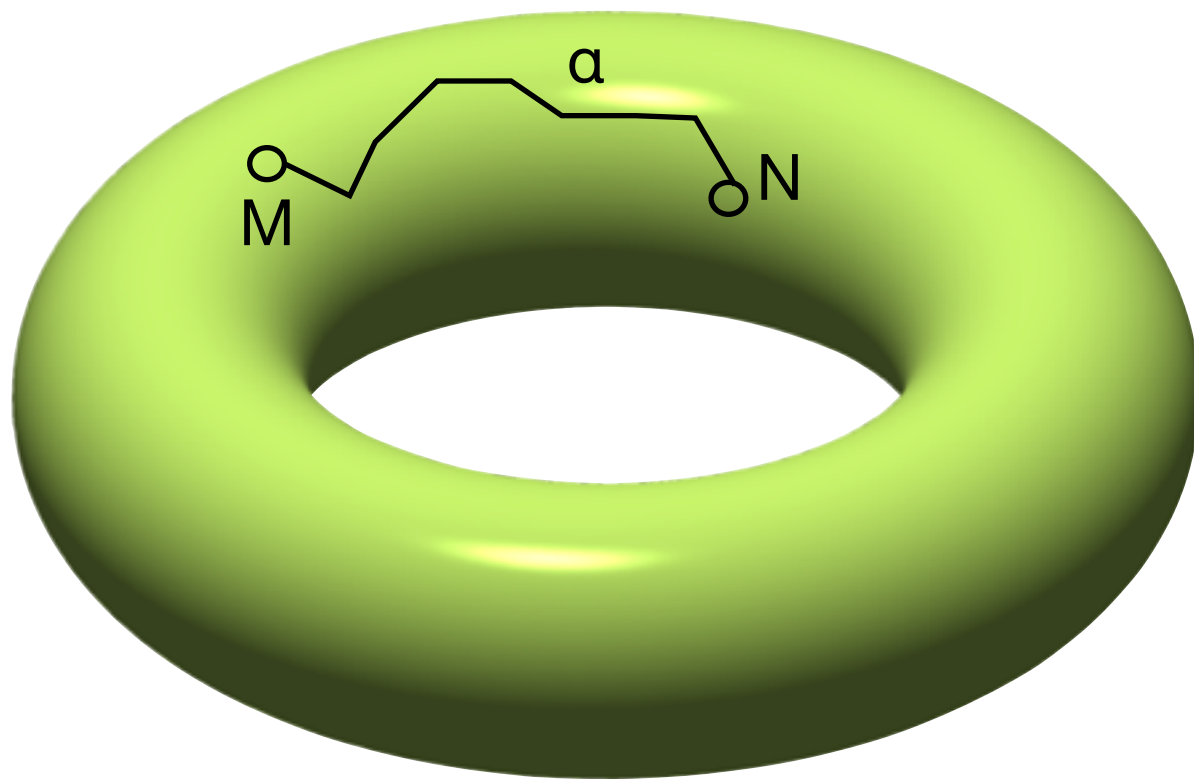
[Hofmann, Streicher, Awodey, Warren, Voevodsky  
Lumsdaine, Gambino, Garner, van den Berg]

# Types as spaces



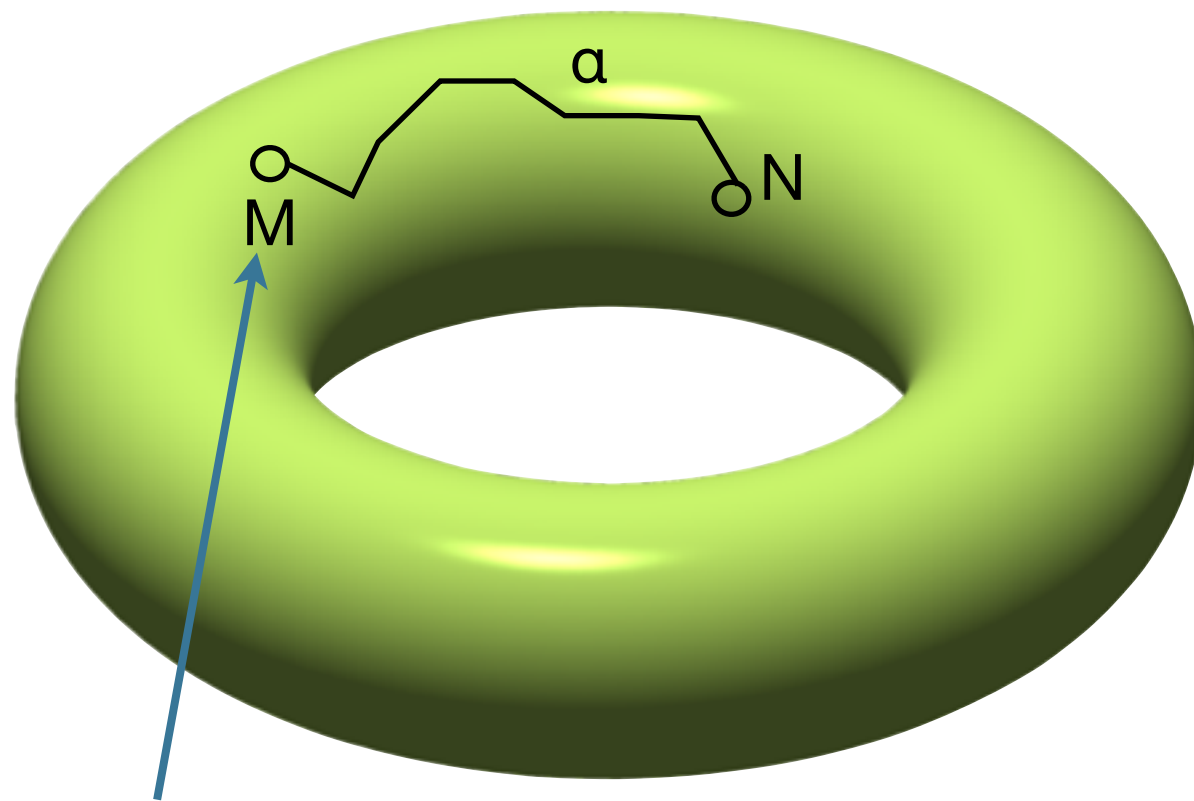
# Types as spaces

**type A is a space**



# Types as spaces

**type A is a space**



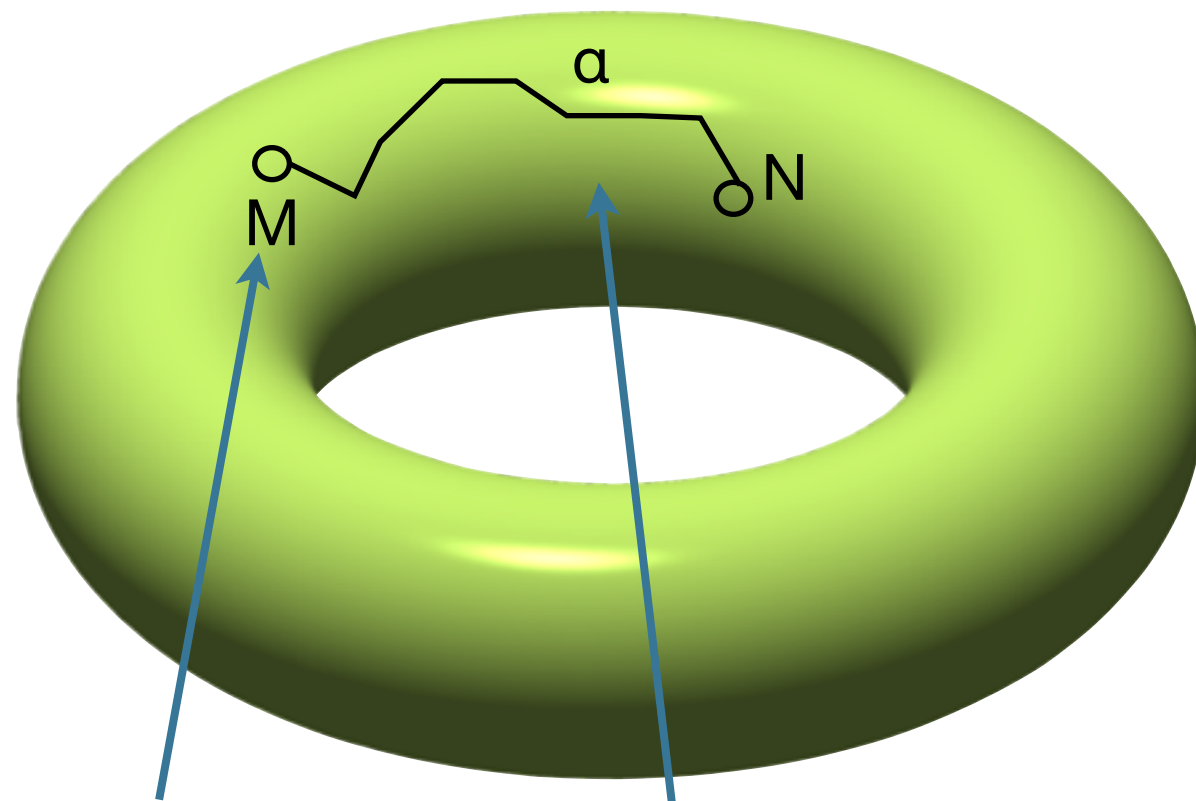
**programs**

$M:A$

**are points**

# Types as spaces

**type A is a space**



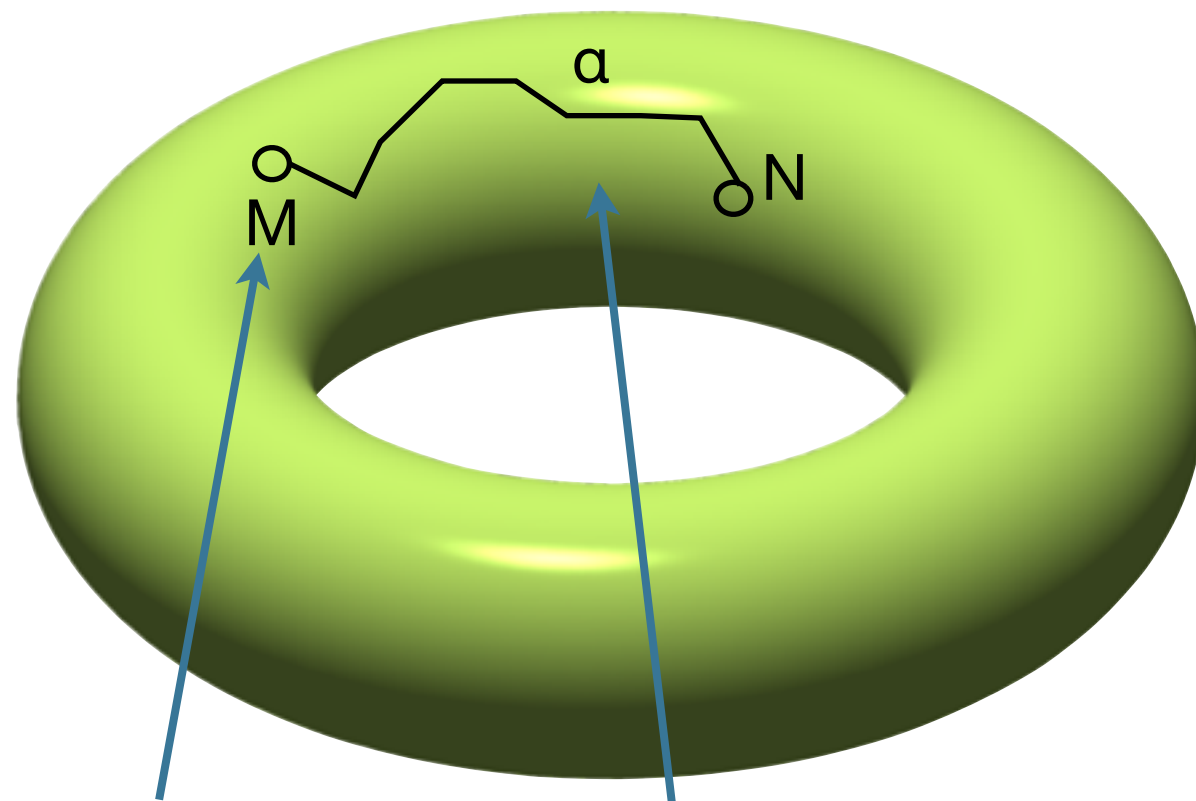
**programs**  
 $M : A$   
**are points**

**proofs of equality**  
 $\alpha : M =_A N$   
**are paths**

# Types as spaces

**type A is a space**

**path operations**



**programs**  
 $M : A$   
**are points**

**proofs of equality**  
 $\alpha : M =_A N$   
**are paths**

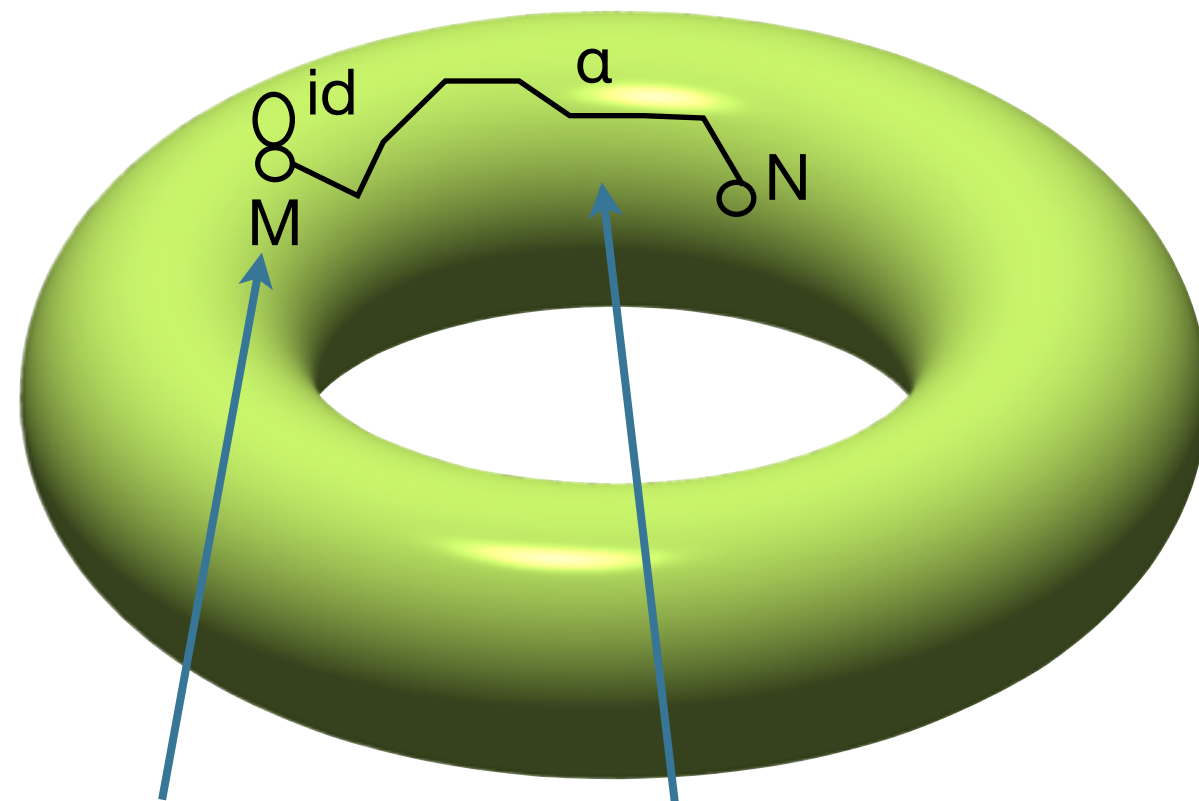


# Types as spaces

**type A is a space**

**path operations**

$\text{id} : M = M \text{ (refl)}$

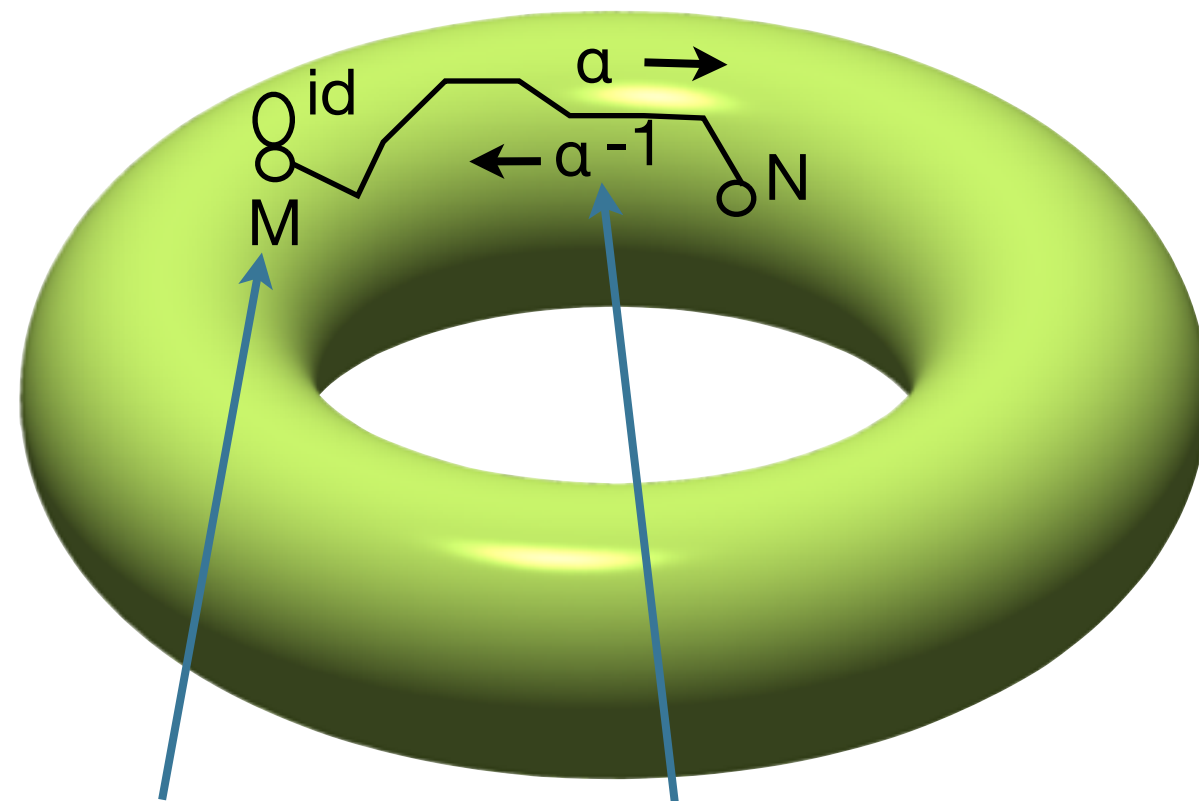


**programs**  
 $M : A$   
**are points**

**proofs of equality**  
 $\alpha : M =_A N$   
**are paths**

# Types as spaces

**type A is a space**



**programs**  
 $M : A$   
**are points**

**proofs of equality**  
 $\alpha : M =_A N$   
**are paths**

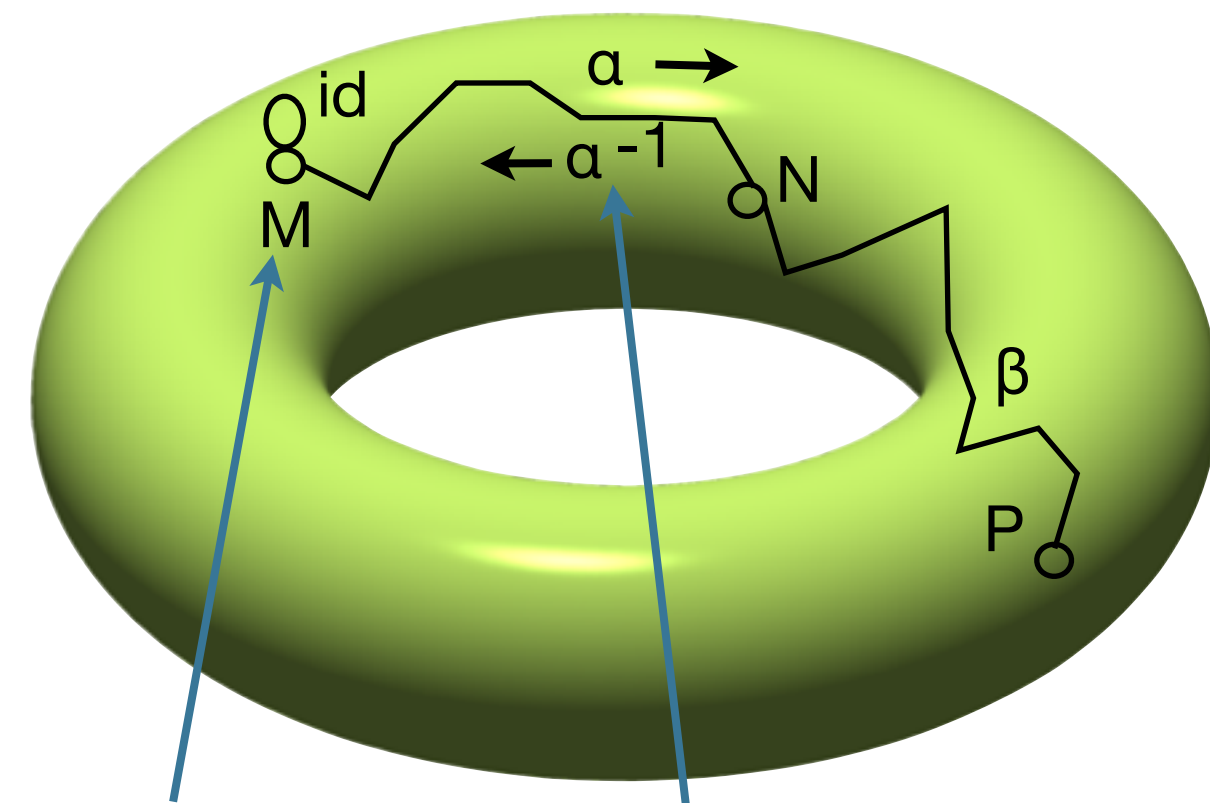
**path operations**

$\text{id} \quad : M = M \text{ (refl)}$

$\alpha^{-1} \quad : N = M \text{ (sym)}$

# Types as spaces

**type A is a space**



**programs**  
 $M:A$   
**are points**

**proofs of equality**  
 $\alpha : M =_A N$   
**are paths**

**path operations**

$\text{id} : M = M \text{ (refl)}$

$\alpha^{-1} : N = M \text{ (sym)}$

$\beta \circ \alpha : M = P \text{ (trans)}$

# Homotopy

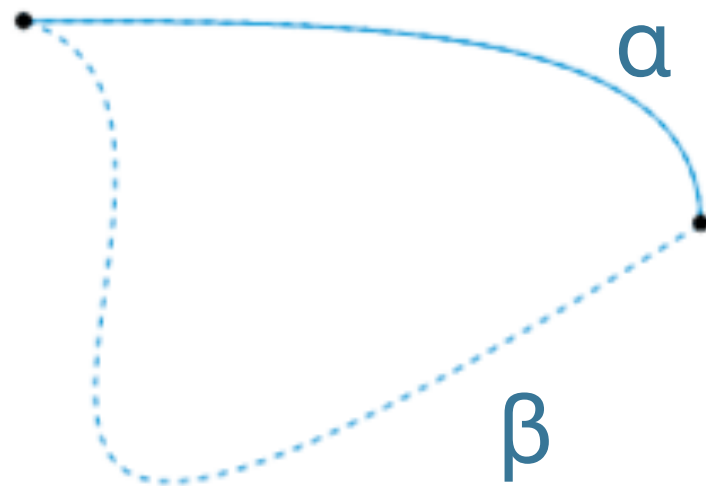
Deformation of one path into another

$\alpha$

$\beta$

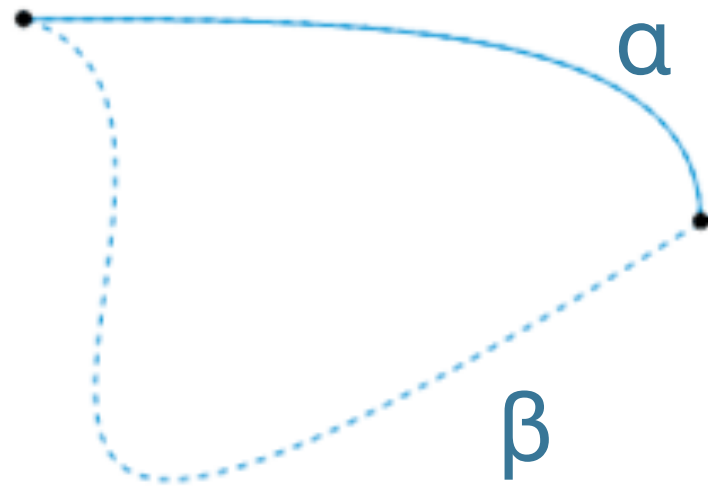
# Homotopy

Deformation of one path into another



# Homotopy

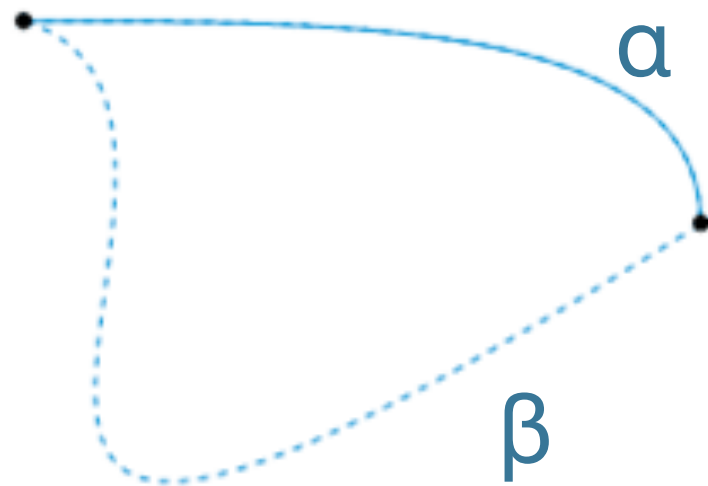
Deformation of one path into another



= 2-dimensional *path between paths*

# Homotopy

Deformation of one path into another

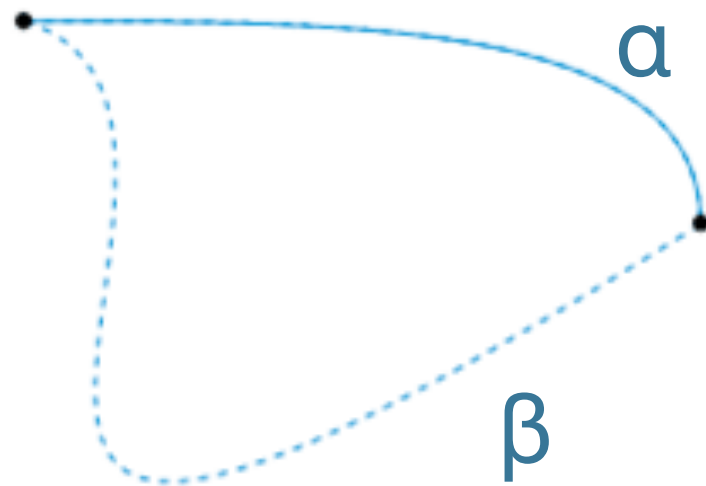


$$\delta : \alpha =_{x=y} \beta$$

= 2-dimensional *path between paths*

# Homotopy

Deformation of one path into another



$$\delta : \alpha =_{x=y} \beta$$

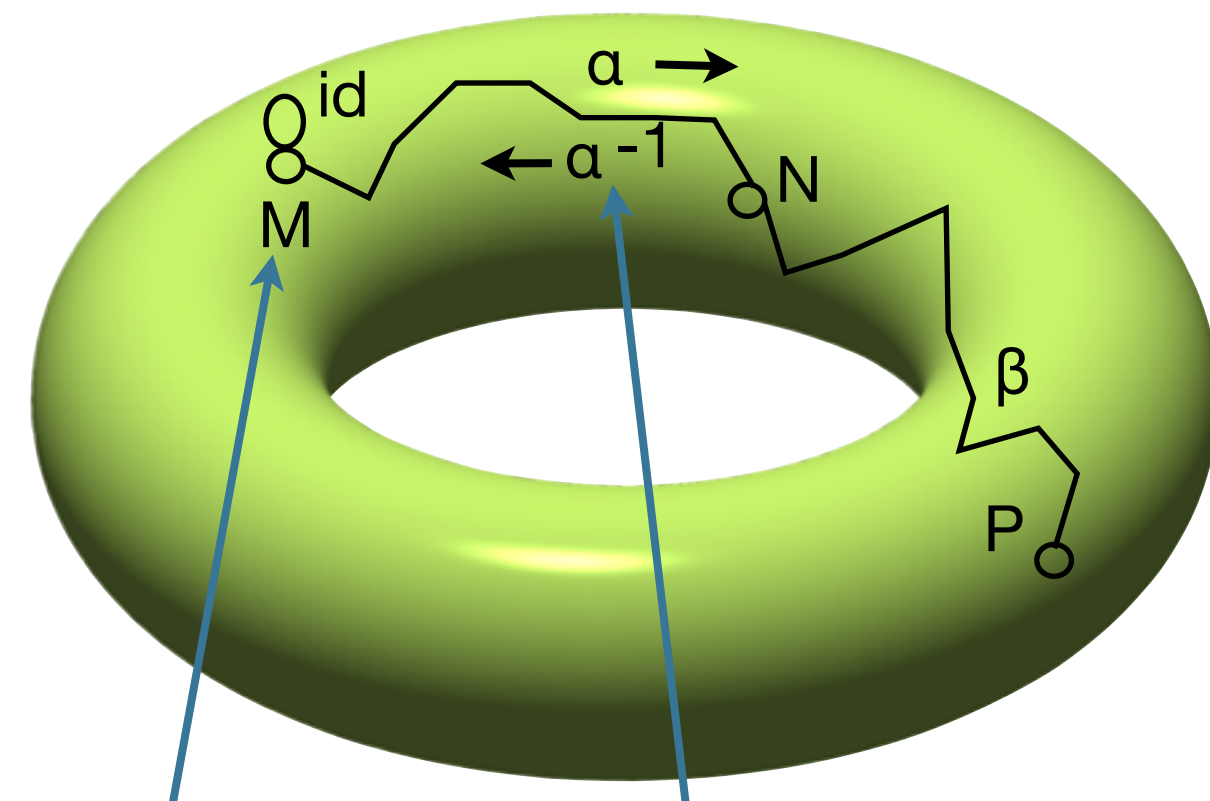
= 2-dimensional *path between paths*

*Then homotopies between homotopies ....*



# Types as spaces

**type A is a space**



**programs**  
 $M:A$   
**are points**

**proofs of equality**  
 $\alpha : M =_A N$   
**are paths**

**path operations**

$\text{id} : M = M \text{ (refl)}$

$\alpha^{-1} : N = M \text{ (sym)}$

$\beta \circ \alpha : M = P \text{ (trans)}$

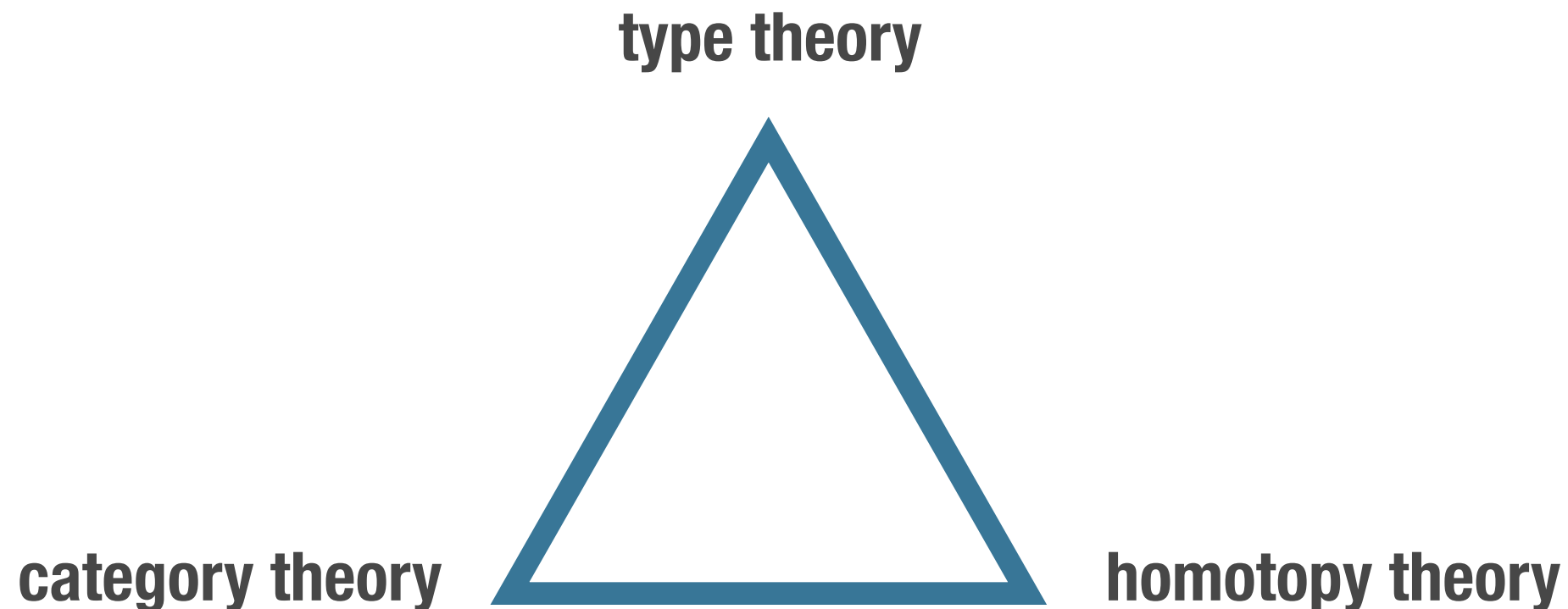
**homotopies**

$\text{ul} : \text{id} \circ \alpha =_{M=N} \alpha$

$\text{il} : \alpha^{-1} \circ \alpha =_{M=M} \text{id}$

$\text{asc} : \gamma \circ (\beta \circ \alpha)$   
 $=_{M=P} (\gamma \circ \beta) \circ \alpha$

# Homotopy Type Theory



[Hofmann, Streicher, Awodey, Warren, Voevodsky  
Lumsdaine, Gambino, Garner, van den Berg]

# Types as $\infty$ -groupoids

**type  $A$  is an  $\infty$ -groupoid**

- \* infinite-dimensional algebraic structure, with morphisms, morphisms between morphisms, ...
- \* each level has a groupoid structure, and they interact

**morphisms**

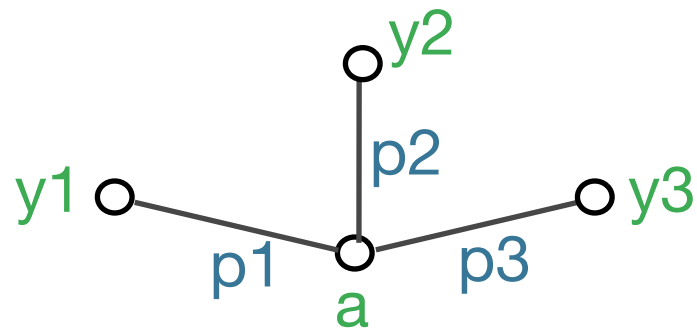
$$\begin{aligned} \text{id} & : M = M \text{ (refl)} \\ \alpha^{-1} & : N = M \text{ (sym)} \\ \beta \circ \alpha & : M = P \text{ (trans)} \end{aligned}$$

**morphisms between morphisms**

$$\begin{aligned} u_l & : \text{id} \circ \alpha =_{M=N} \alpha \\ i_l & : \alpha^{-1} \circ \alpha =_{M=M} \text{id} \\ \text{asc} & : \gamma \circ (\beta \circ \alpha) \\ & \quad =_{M=P} (\gamma \circ \beta) \circ \alpha \end{aligned}$$

# Path induction

**Type of paths  
from  $a$  to somewhere**



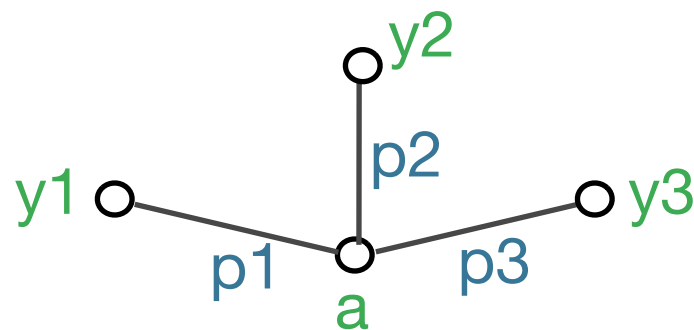
**is inductively  
generated by**



# Path induction

Type of paths  
from  $a$  to somewhere

is inductively  
generated by



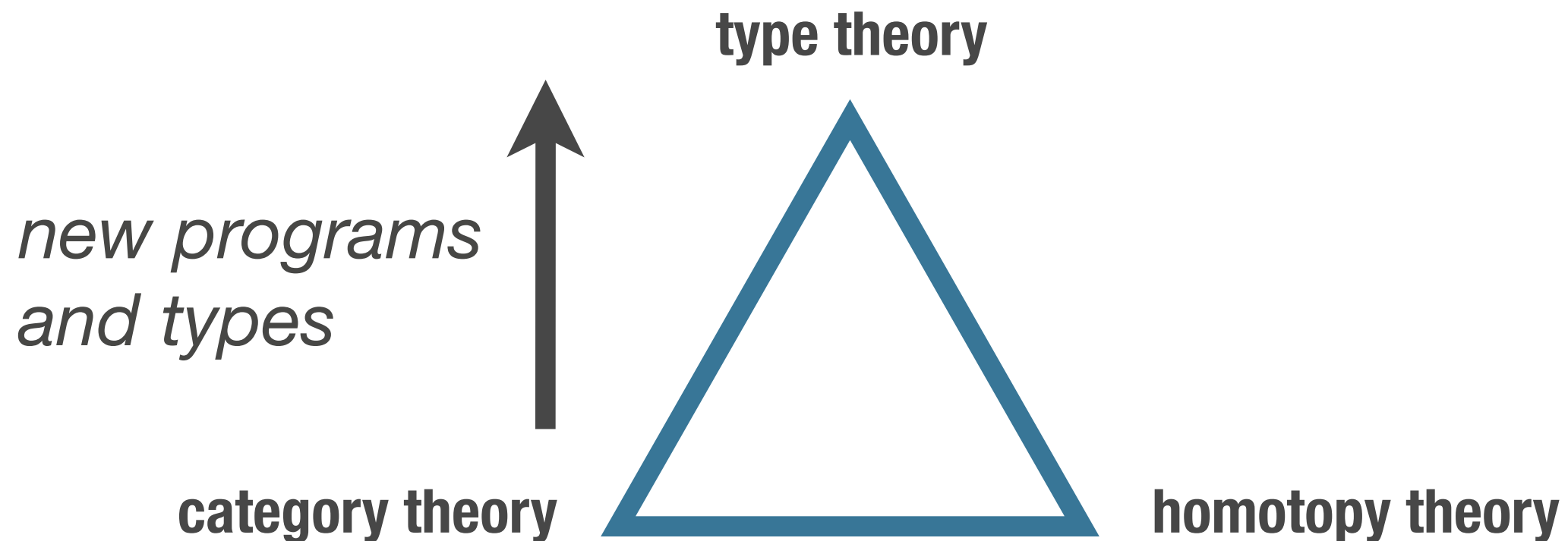
Fix a type  $A$  with element  $a:A$ .

For a family of types  $C(y:A, p:a=y)$ ,  
to give an element of

$C(y, p)$  for all  $y$  and  $p:a=y$ ,  
suffices to give an element of  
 $C(a, id)$

Type theory is a  
synthetic theory of  
spaces/ $\infty$ -groupoids

# Homotopy Type Theory



# Univalence [Voevodsky]



# Univalence [Voevodsky]

- ✱ *Equivalence of types* is a generalization to spaces of bijection of sets

# Univalence [Voevodsky]

- ✱ *Equivalence of types* is a generalization to spaces of bijection of sets
- ✱ Univalence axiom:  
equality of types ( $A =_{\text{Type}} B$ ) is (equivalent to)  
equivalence of types ( $\text{Equiv } A \ B$ )

# Univalence [Voevodsky]

- \* *Equivalence of types* is a generalization to spaces of bijection of sets
- \* Univalence axiom:  
equality of types ( $A =_{\text{Type}} B$ ) is (equivalent to)  
equivalence of types ( $\text{Equiv } A \ B$ )
- \*  $\therefore$  all structures/properties respect equivalence

# Univalence [Voevodsky]

- \* *Equivalence of types* is a generalization to spaces of bijection of sets
- \* Univalence axiom:  
equality of types ( $A =_{\text{Type}} B$ ) is (equivalent to)  
equivalence of types ( $\text{Equiv } A \ B$ )
- \*  $\therefore$  all structures/properties respect equivalence
- \* Not by collapsing equivalence,  
but by exploiting proof-relevant equality:  
*transport does real work*

# Higher inductive types

[Bauer,Lumsdaine,Shulman,Warren]

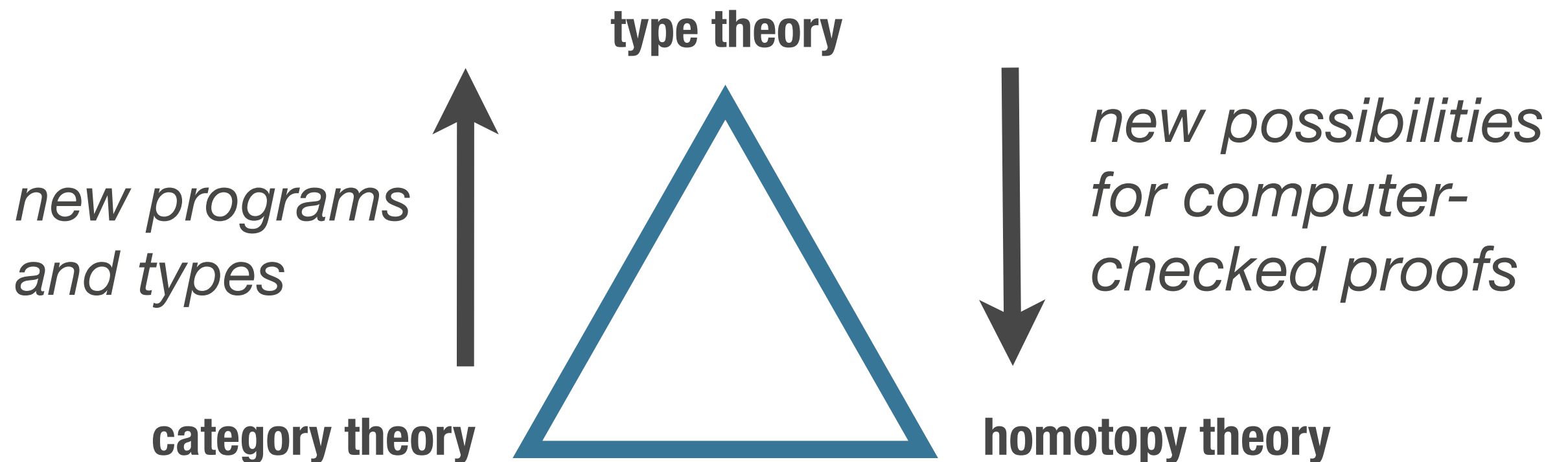
*New way of forming types:*

Inductive type specified by generators  
not only for points (elements), but also for paths

# Constructivity

- \* Non-affirmation of classical principles ✓
- \* Computational interpretation ?
- \* Proof-relevant mathematics ✓

# Homotopy Type Theory



# Outline

1. Certified homotopy theory
2. Certified software



# Outline

**1.Certified homotopy theory**

2.Certified software

# Homotopy Theory

A branch of topology,  
the study of spaces and continuous deformations



[image from wikipedia]

# Homotopy in HoTT

$$\pi_1(S^1) = \mathbb{Z}$$

Freudenthal

Van Kampen

$$\pi_{k < n}(S^n) = 0$$

$$\pi_n(S^n) = \mathbb{Z}$$

Covering spaces

Hopf fibration

$$K(G, n)$$

Whitehead  
for n-types

$$\pi_2(S^2) = \mathbb{Z}$$

Cohomology  
axioms

$$\pi_3(S^2) = \mathbb{Z}$$

Blakers-Massey

James

Construction

$$\pi_4(S^3) = \mathbb{Z}?$$

**[Brunerie, Finster, Hou,  
Licata, Lumsdaine, Shulman]**

# Homotopy in HoTT


$$\pi_1(S^1) = \mathbb{Z}$$

$$\pi_{k < n}(S^n) = 0$$

Hopf fibration

$$\pi_2(S^2) = \mathbb{Z}$$

$$\pi_3(S^2) = \mathbb{Z}$$

James  
Construction

$$\pi_4(S^3) = \mathbb{Z}?$$

Freudenthal

$$\pi_n(S^n) = \mathbb{Z}$$

$K(G, n)$

Cohomology  
axioms

Blakers-Massey

Van Kampen

Covering spaces

Whitehead  
for n-types

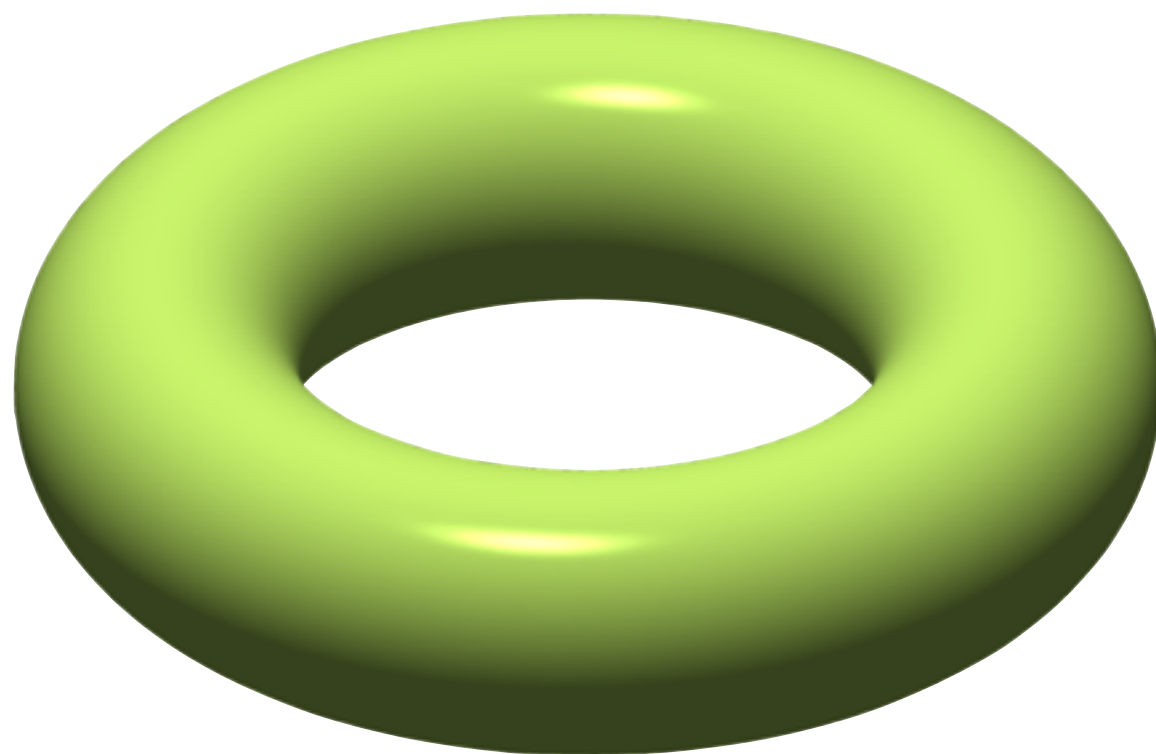
**[Brunerie, Finster, Hou,  
Licata, Lumsdaine, Shulman]**

# Homotopy Groups

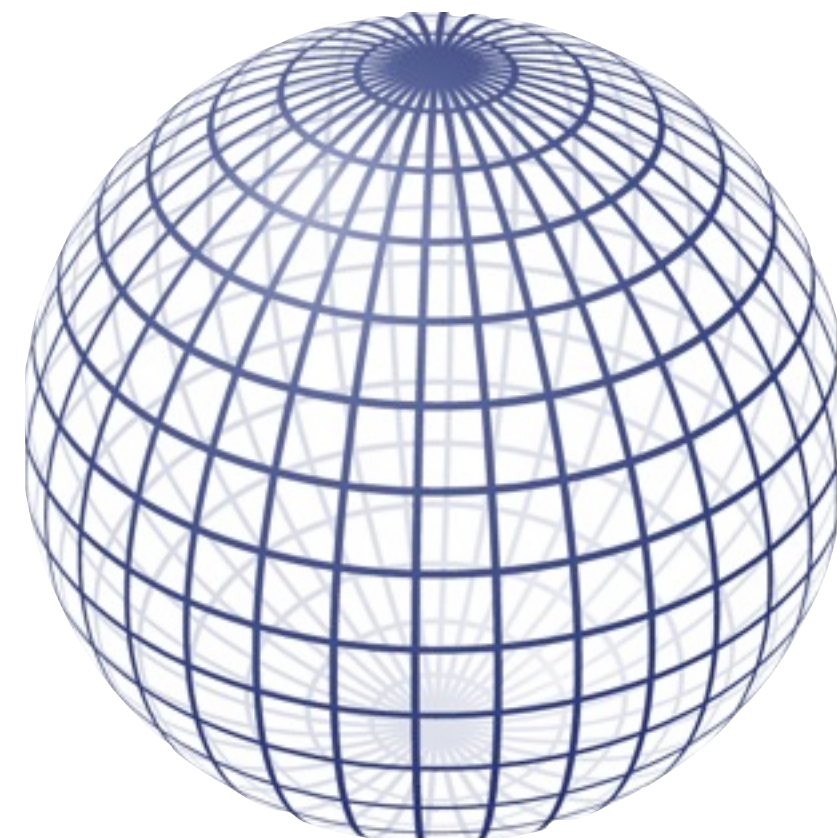
*Homotopy groups of a space  $X$ :*

- \*  $\pi_1(X)$  is fundamental group (group of loops)
- \*  $\pi_2(X)$  is group of homotopies (2-dimensional loops)
- \*  $\pi_3(X)$  is group of 3-dimensional loops
- \* ...

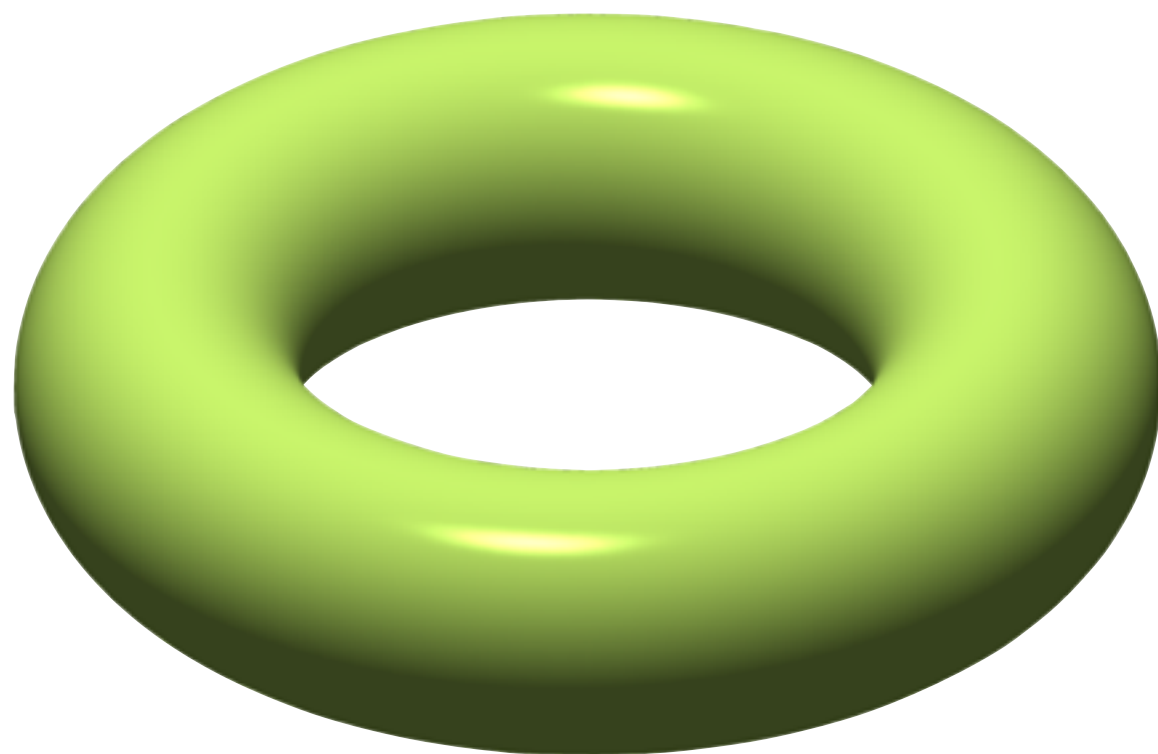
# Telling spaces apart



$\neq$

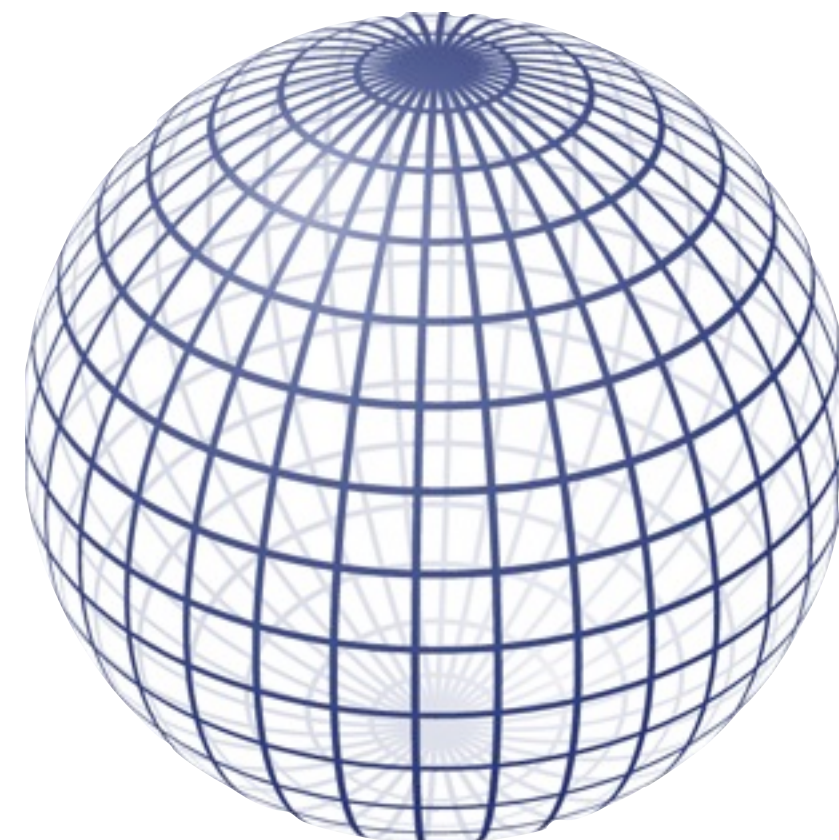


# Telling spaces apart



fundamental group  
is non-trivial ( $\mathbb{Z} \times \mathbb{Z}$ )

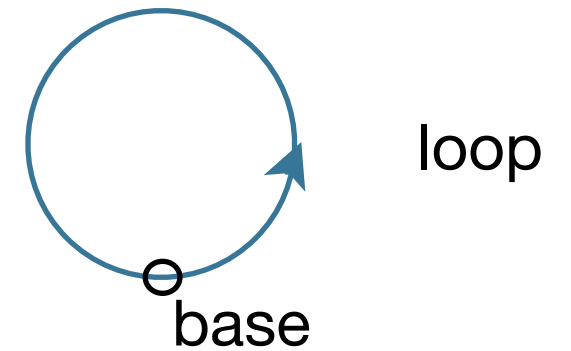
$\neq$



fundamental group  
is trivial

# The Circle

Circle  $S^1$  is a **higher inductive type** generated by



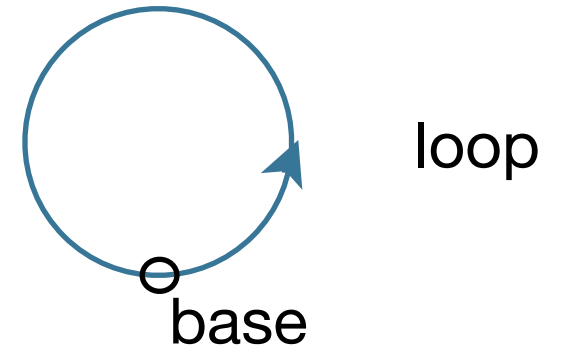


# The Circle

Circle  $S^1$  is a **higher inductive type** generated by

base :  $S^1$

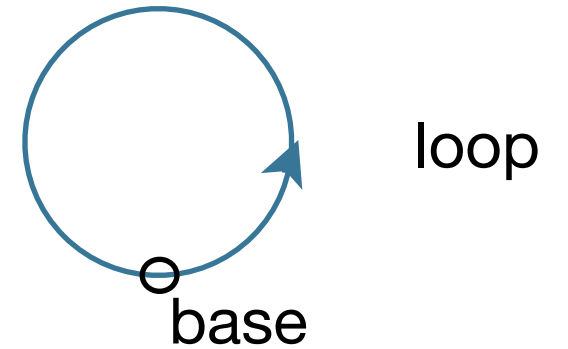
loop : base = base



# The Circle

Circle  $S^1$  is a **higher inductive type** generated by

**point**     $\text{base} : S^1$   
           $\text{loop} : \text{base} = \text{base}$

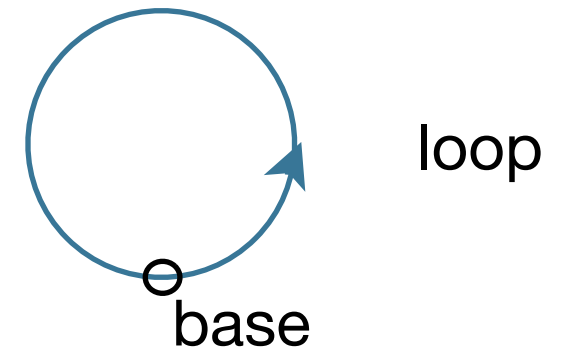


# The Circle

Circle  $S^1$  is a **higher inductive type** generated by

**point**     $\text{base} : S^1$

**path**     $\text{loop} : \text{base} = \text{base}$

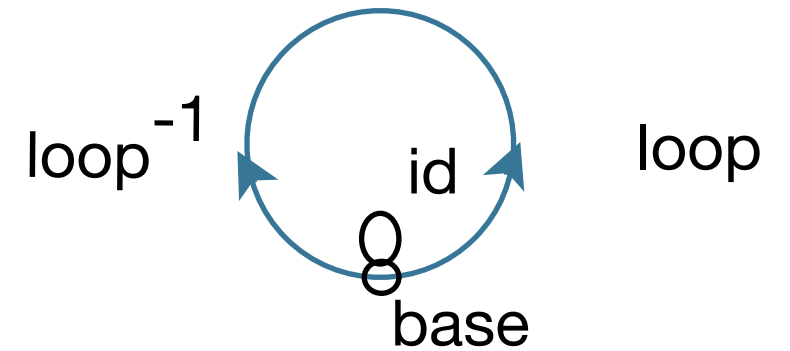


# The Circle

Circle  $S^1$  is a **higher inductive type** generated by

**point**     $\text{base} : S^1$

**path**     $\text{loop} : \text{base} = \text{base}$



*Free type:* equipped with structure

$\text{id}$                        $\text{inv} : \text{loop} \circ \text{loop}^{-1} = \text{id}$

$\text{loop}^{-1}$                        $\dots$

$\text{loop} \circ \text{loop}$

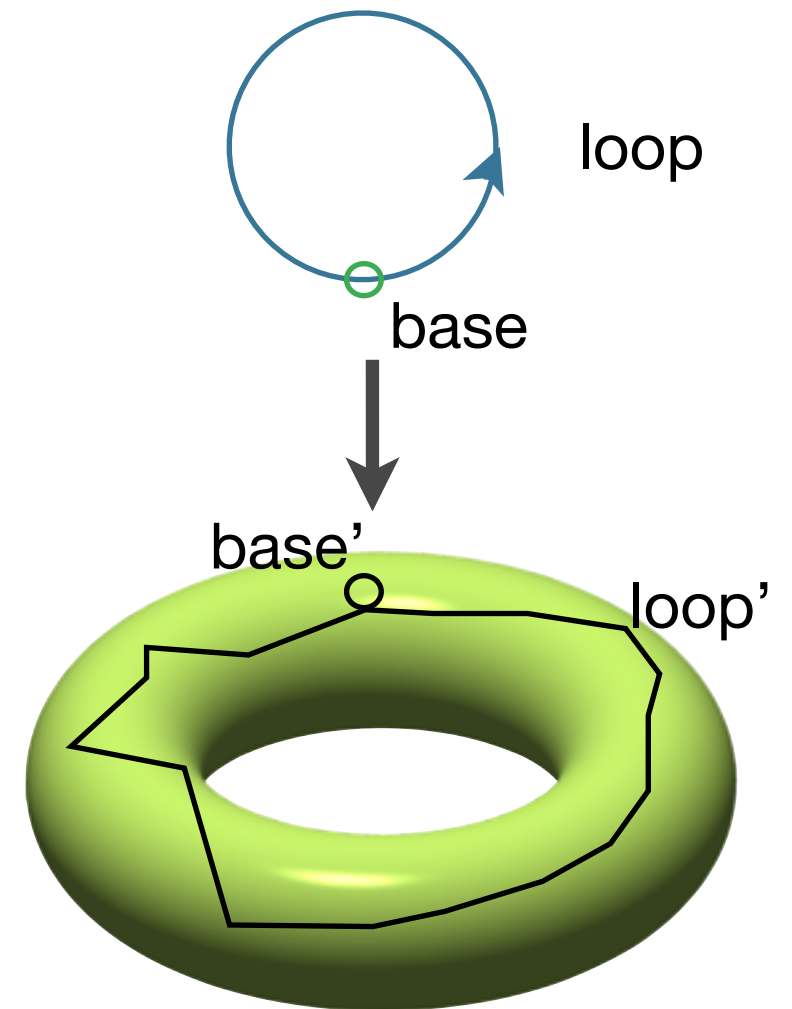
# The Circle

## Circle recursion:

function  $S^1 \rightarrow X$  determined by

$\text{base}' : X$

$\text{loop}' : \text{base}' = \text{base}'$



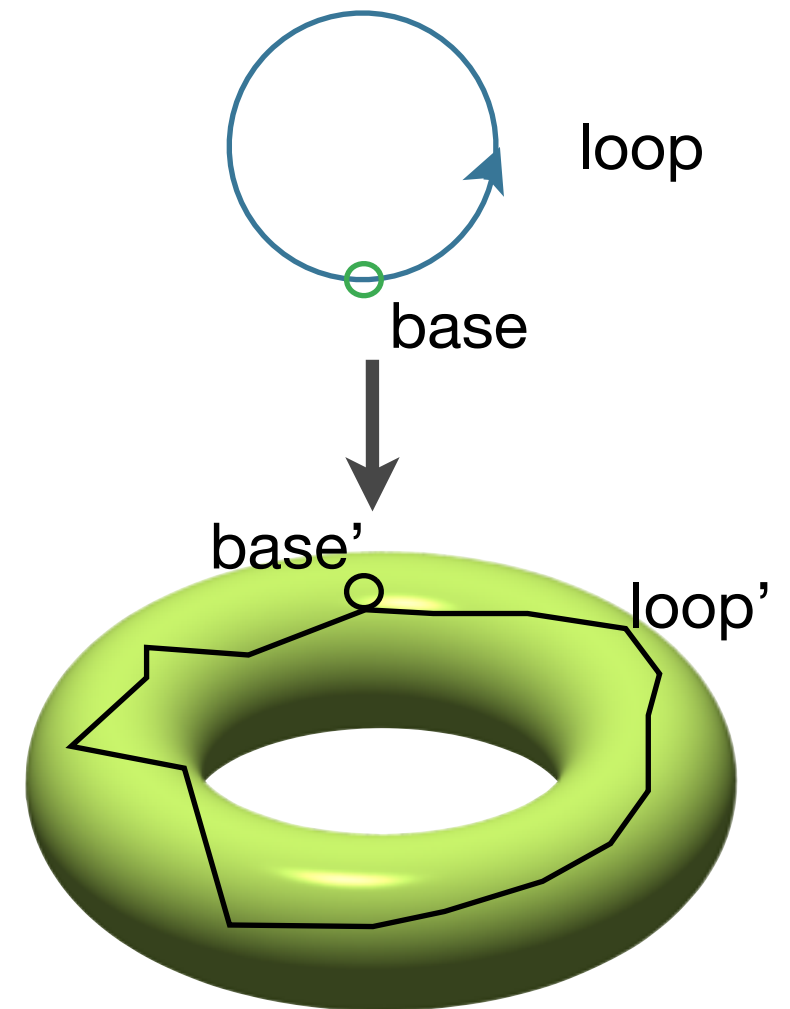
# The Circle

## Circle recursion:

function  $S^1 \rightarrow X$  determined by

$\text{base}' : X$

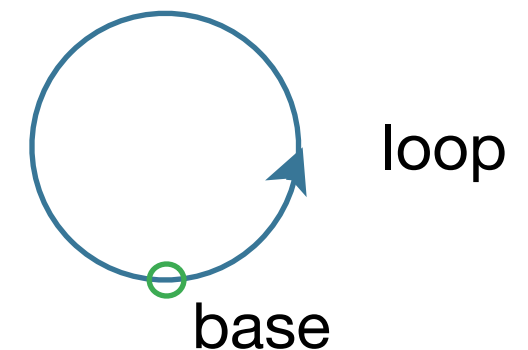
$\text{loop}' : \text{base}' = \text{base}'$



**Circle induction:** To prove a predicate  $P$  for all points on the circle, suffices to prove  $P(\text{base})$ , continuously in the loop

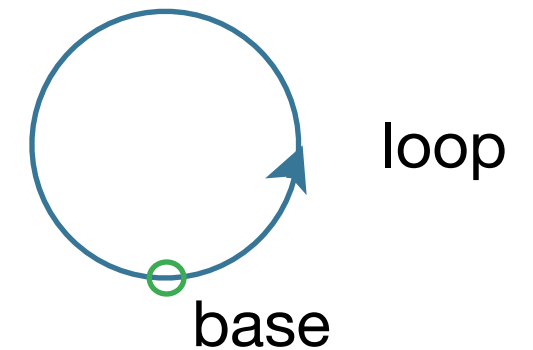
# Fundamental group of circle

How many different loops are there on the circle, up to homotopy?



# Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

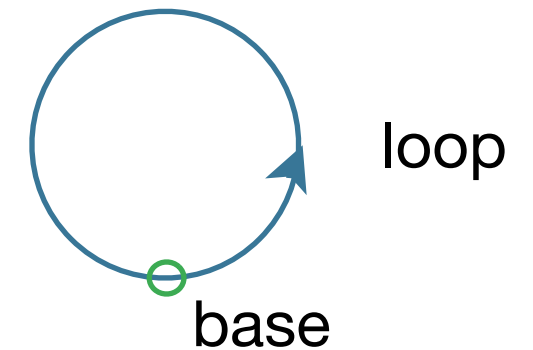


id



# Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

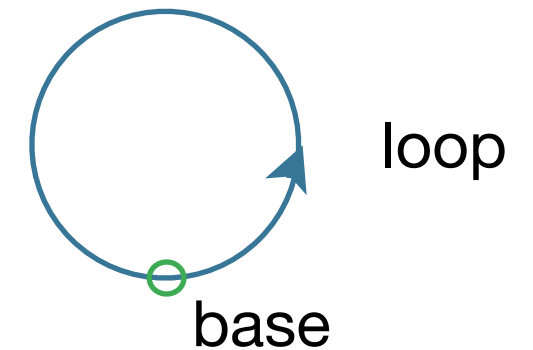


id

loop

# Fundamental group of circle

How many different loops are there on the circle, up to homotopy?



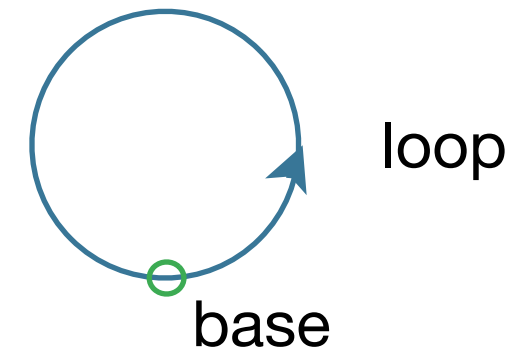
$\text{id}$

$\text{loop}$

$\text{loop}^{-1}$

# Fundamental group of circle

How many different loops are there on the circle, up to homotopy?



$\text{id}$

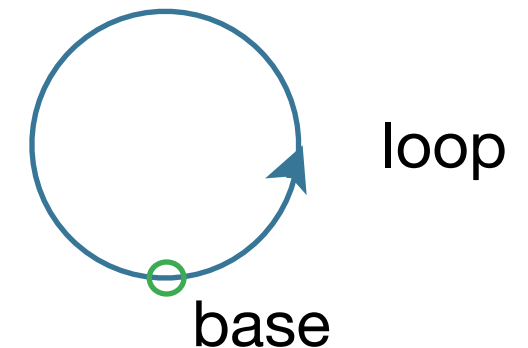
$\text{loop}$

$\text{loop}^{-1}$

$\text{loop} \circ \text{loop}$

# Fundamental group of circle

How many different loops are there on the circle, up to homotopy?



$\text{id}$

$\text{loop}$

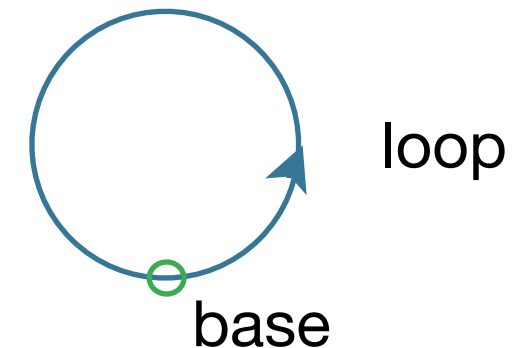
$\text{loop}^{-1}$

$\text{loop} \circ \text{loop}$

$\text{loop}^{-1} \circ \text{loop}^{-1}$

# Fundamental group of circle

How many different loops are there on the circle, up to homotopy?



id

loop

$\text{loop}^{-1}$

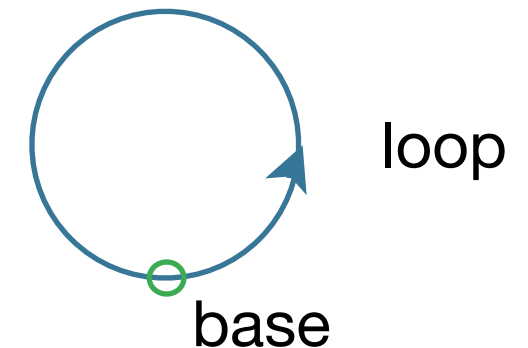
$\text{loop} \circ \text{loop}$

$\text{loop}^{-1} \circ \text{loop}^{-1}$

$\text{loop} \circ \text{loop}^{-1}$

# Fundamental group of circle

How many different loops are there on the circle, up to homotopy?



$\text{id}$

$\text{loop}$

$\text{loop}^{-1}$

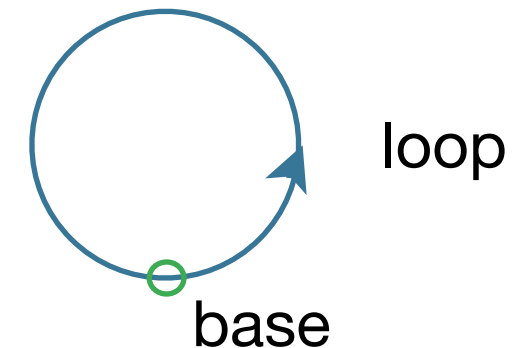
$\text{loop} \circ \text{loop}$

$\text{loop}^{-1} \circ \text{loop}^{-1}$

$\text{loop} \circ \text{loop}^{-1} = \text{id}$

# Fundamental group of circle

How many different loops are there on the circle, up to homotopy?



$\text{id}$   $\emptyset$

$\text{loop}$

$\text{loop}^{-1}$

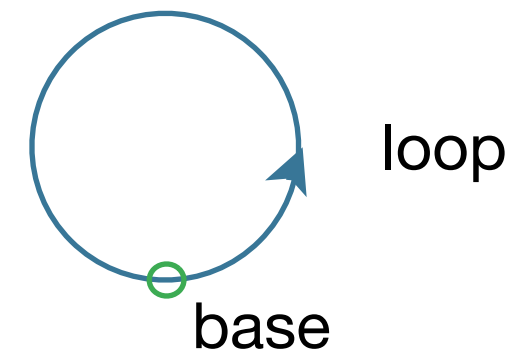
$\text{loop} \circ \text{loop}$

$\text{loop}^{-1} \circ \text{loop}^{-1}$

$\text{loop} \circ \text{loop}^{-1} = \text{id}$

# Fundamental group of circle

# How many different loops are there on the circle, up to homotopy?



id 0

loop 1

$$\text{loop}^{-1}$$

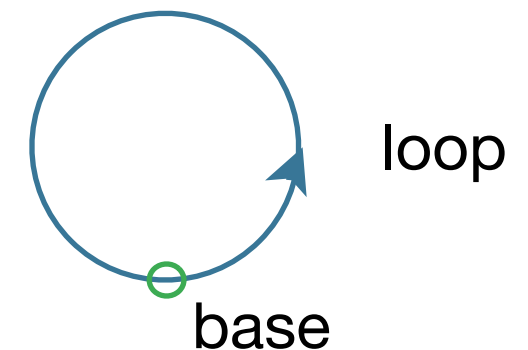
loop o loop

$$\text{loop}^{-1} \circ \text{loop}^{-1}$$
$$\text{loop} \circ \text{loop}^{-1} = \text{id}$$



# Fundamental group of circle

# How many different loops are there on the circle, up to homotopy?



id 0

loop 1

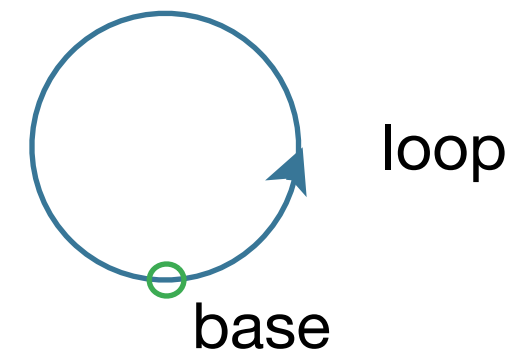
$$\text{loop}^{-1} \quad -1$$

loop o loop

$$\text{loop}^{-1} \circ \text{loop}^{-1}$$
$$\text{loop} \circ \text{loop}^{-1} = \text{id}$$

# Fundamental group of circle

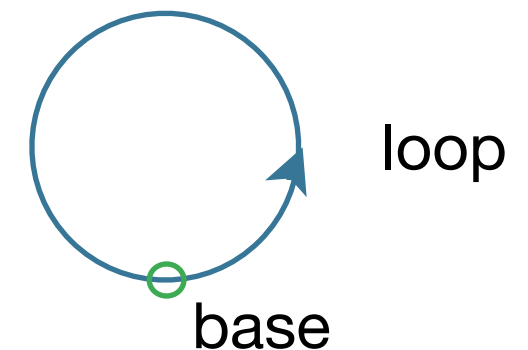
How many different loops are there on the circle, up to homotopy?



id	0
loop	1
loop <sup>-1</sup>	-1
loop o loop	2
loop <sup>-1</sup> o loop <sup>-1</sup>	
loop o loop <sup>-1</sup>	= id

# Fundamental group of circle

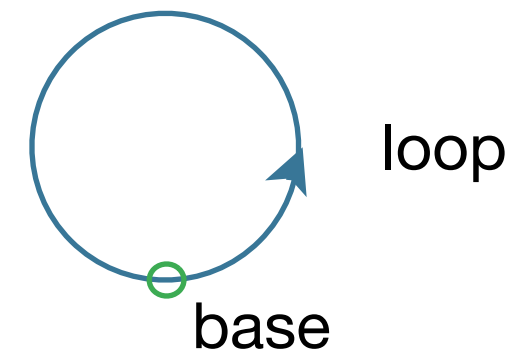
How many different loops are there on the circle, up to homotopy?



id	0
loop	1
loop <sup>-1</sup>	-1
loop o loop	2
loop <sup>-1</sup> o loop <sup>-1</sup>	-2
loop o loop <sup>-1</sup>	= id

# Fundamental group of circle

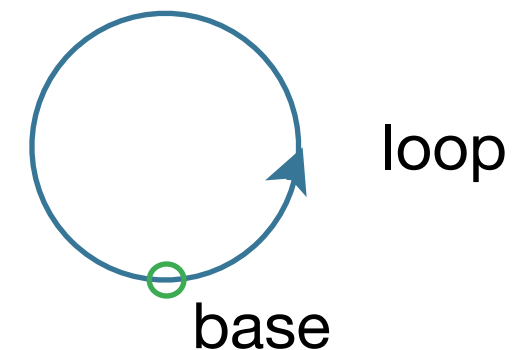
How many different loops are there on the circle, up to homotopy?



$\text{id}$	$0$
$\text{loop}$	$1$
$\text{loop}^{-1}$	$-1$
$\text{loop} \circ \text{loop}$	$2$
$\text{loop}^{-1} \circ \text{loop}^{-1}$	$-2$
$\text{loop} \circ \text{loop}^{-1} = \text{id}$	$0$

# Fundamental group of circle

How many different loops are there on the circle, up to homotopy?



id	0
----	---

loop	1
------	---

$\text{loop}^{-1}$	-1
--------------------	----

$\text{loop} \circ \text{loop}$	2
---------------------------------	---

$\text{loop}^{-1} \circ \text{loop}^{-1}$	-2
---	----

$\text{loop} \circ \text{loop}^{-1} = \text{id}$	0
--	---

**integers are “codes”  
for paths on the  
circle**

# Fundamental group of circle

**Definition.**  $\Omega(S^1)$  is the **type** of loops at base  
i.e. the type  $(\text{base} =_{S^1} \text{base})$

# Fundamental group of circle

**Definition.**  $\Omega(S^1)$  is the **type** of loops at base  
i.e. the type  $(\text{base} =_{S^1} \text{base})$

**Theorem.**  $\Omega(S^1)$  is equivalent to  $\mathbb{Z}$ ,  
by a map that sends 0 to +

# Fundamental group of circle

**Definition.**  $\Omega(S^1)$  is the **type** of loops at base  
i.e. the type  $(\text{base} =_{S^1} \text{base})$

**Theorem.**  $\Omega(S^1)$  is equivalent to  $\mathbb{Z}$ ,  
by a map that sends 0 to +

**Corollary:** Fundamental group  
of the circle is isomorphic to  $\mathbb{Z}$



# Fundamental group of circle

**Definition.**  $\Omega(S^1)$  is the **type** of loops at base  
i.e. the type  $(\text{base} =_{S^1} \text{base})$

**Theorem.**  $\Omega(S^1)$  is equivalent to  $\mathbb{Z}$ ,  
by a map that sends 0 to +

**Corollary:** Fundamental group  
of the circle is isomorphic to  $\mathbb{Z}$

**0-truncation (set of connected components)**  
of  $\Omega(S^1)$



# Fundamental group of circle

**Theorem.**  $\Omega(S^1)$  is equivalent to  $\mathbb{Z}$

**Proof (Shulman, L.):** two mutually inverse functions

$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

$$\text{loop}^- : \mathbb{Z} \rightarrow \Omega(S^1)$$

# Fundamental group of circle

**Theorem.**  $\Omega(S^1)$  is equivalent to  $\mathbb{Z}$

**Proof (Shulman, L.):** two mutually inverse functions

$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

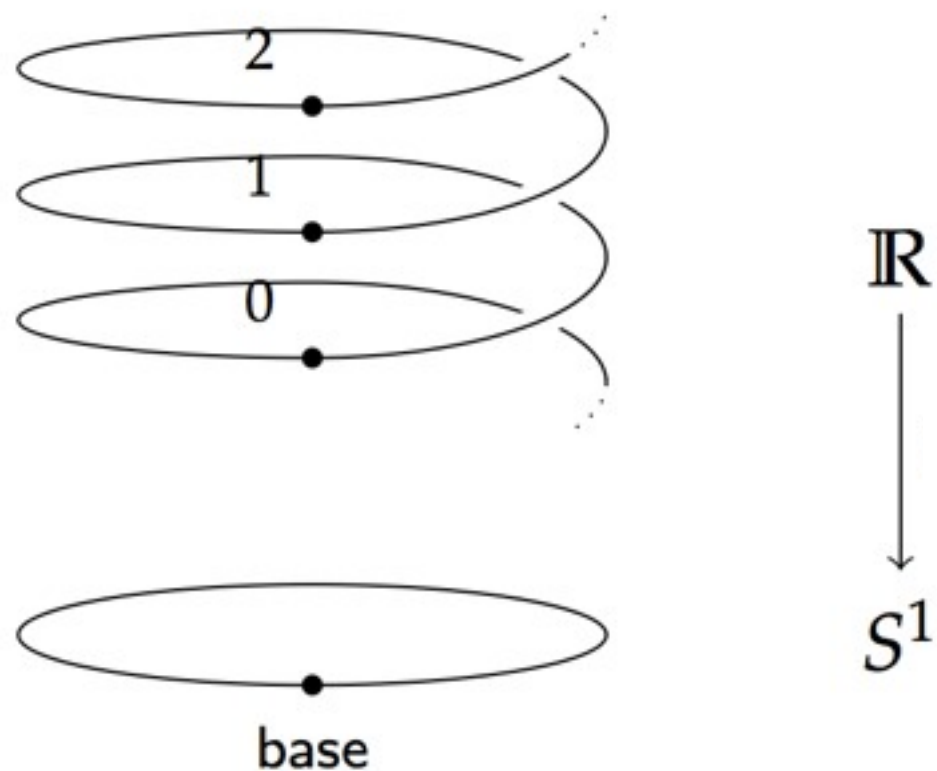
$$\text{loop}^- : \mathbb{Z} \rightarrow \Omega(S^1)$$

$$\text{loop}^0 = \text{id}$$

$$\text{loop}^{+n} = \text{loop} \circ \text{loop} \circ \dots \circ \text{loop} \quad (n \text{ times})$$

$$\text{loop}^{-n} = \text{loop}^{-1} \circ \text{loop}^{-1} \circ \dots \circ \text{loop}^{-1} \quad (n \text{ times})$$

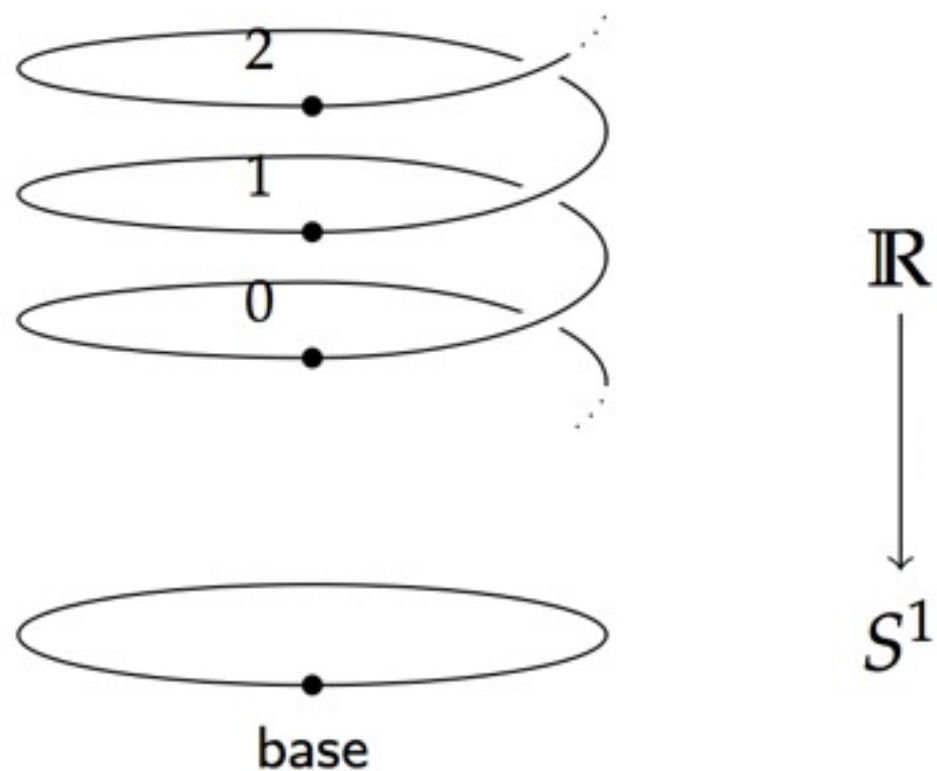
# Universal Cover



$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

defined by **lifting** a loop to the cover, and giving the other endpoint of 0

# Universal Cover

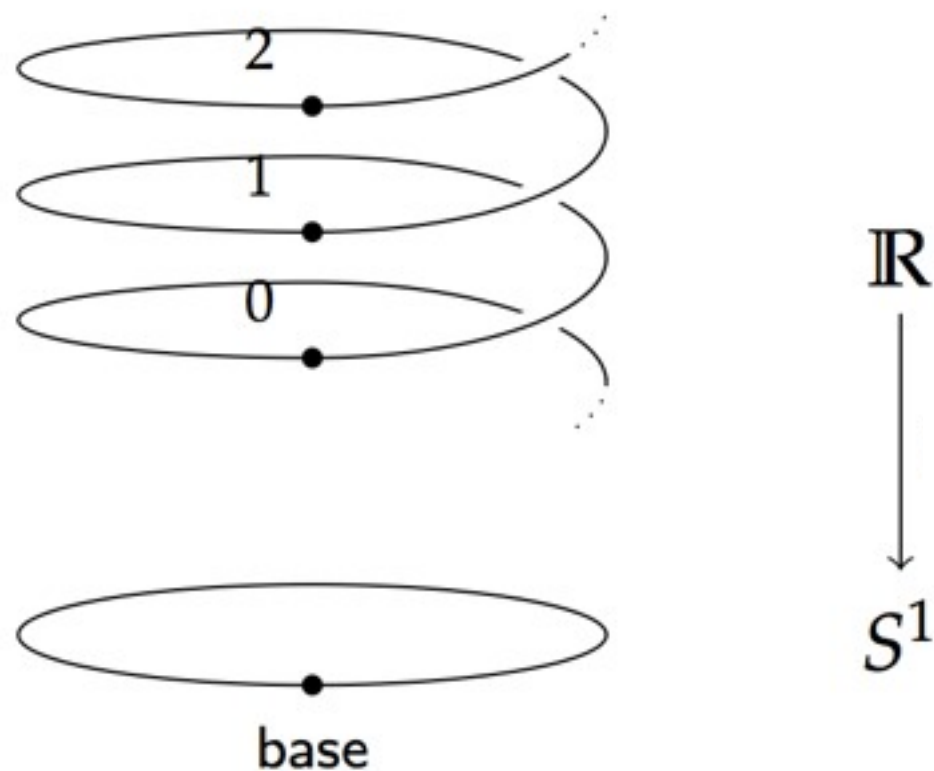


$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

defined by **lifting** a loop to the cover, and giving the other endpoint of 0

lifting is functorial

# Universal Cover



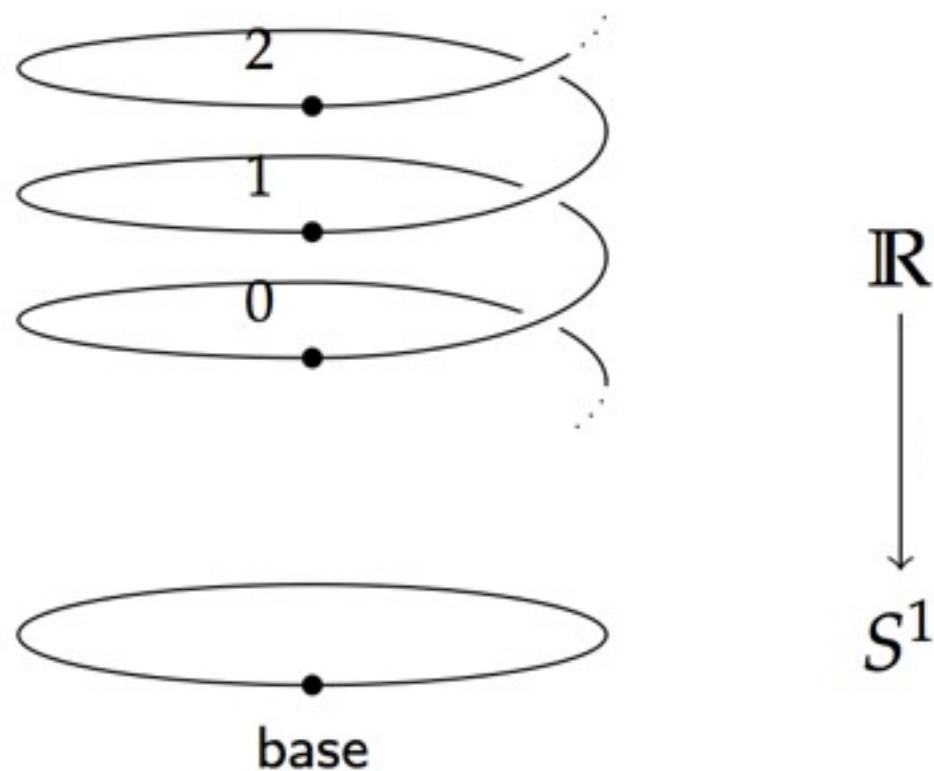
$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

defined by **lifting** a loop to the cover, and giving the other endpoint of 0

lifting is functorial

lifting loop adds 1

# Universal Cover



$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

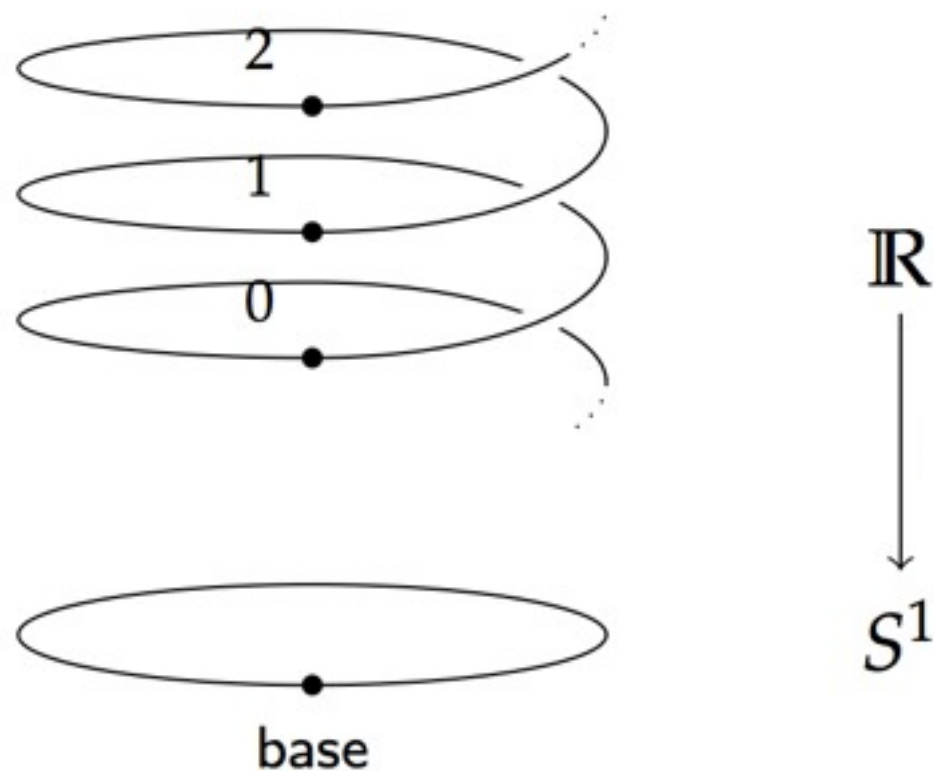
defined by **lifting** a loop to the cover, and giving the other endpoint of 0

lifting is functorial

lifting  $\text{loop}$  adds 1

lifting  $\text{loop}^{-1}$  subtracts 1

# Universal Cover



$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

defined by **lifting** a loop to the cover, and giving the other endpoint of 0

**Example:**

$$\begin{aligned} & \text{wind}(\text{loop} \circ \text{loop}^{-1}) \\ &= 0 + 1 - 1 \\ &= 0 \end{aligned}$$

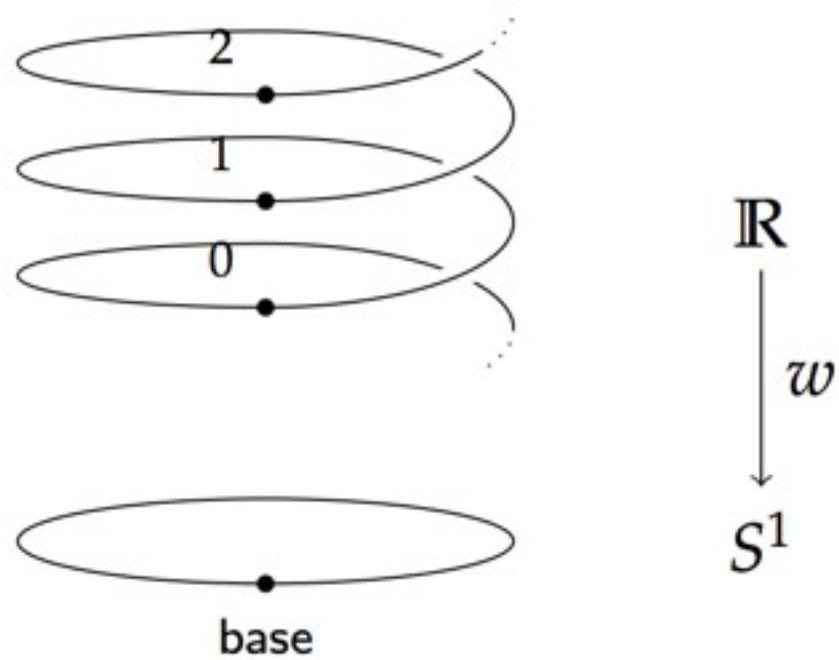
lifting is functorial

lifting  $\text{loop}$  adds 1

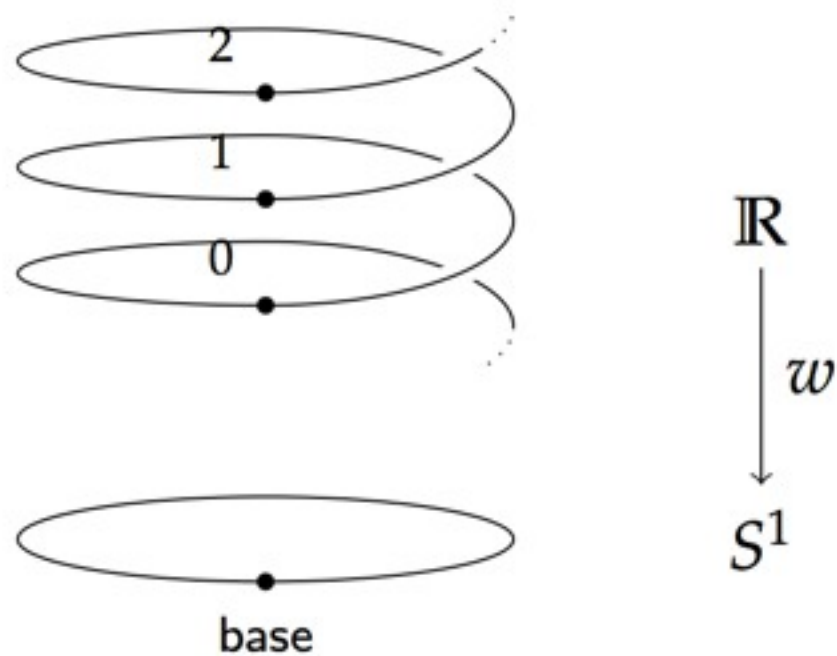
lifting  $\text{loop}^{-1}$  subtracts 1



# Universal Cover



# Universal Cover



$\text{Cover} : S^1 \rightarrow \text{Type}$

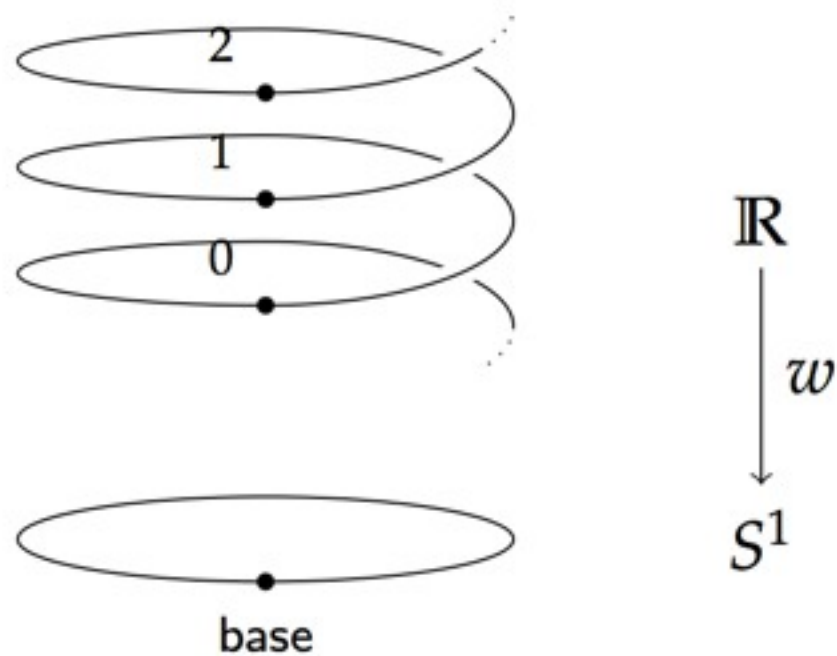
$\text{Cover}(\text{base}) := \mathbb{Z}$

$\text{Cover}_1(\text{loop}) :=$

$\text{ua}(\text{successor}) : \mathbb{Z} = \mathbb{Z}$

# Universal Cover

defined by circle  
recursion

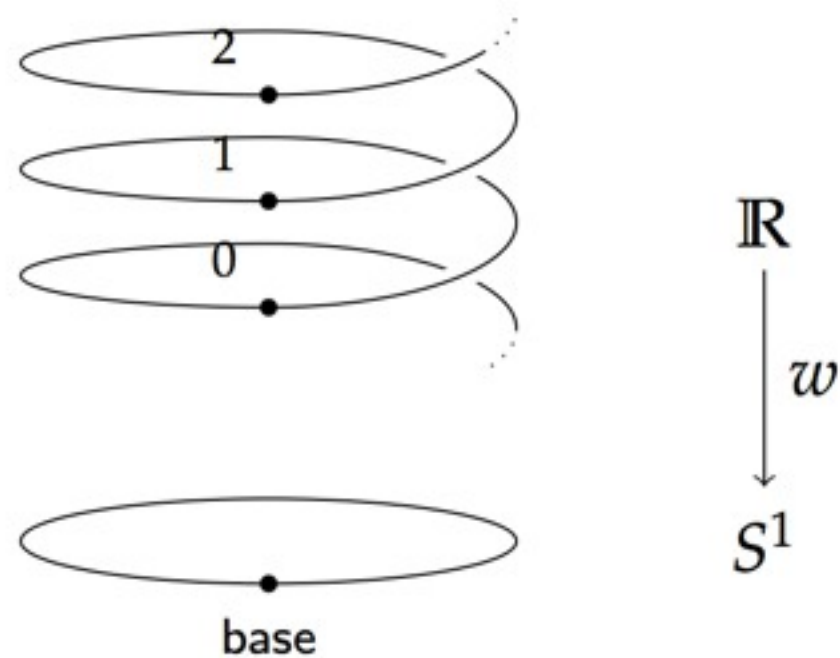


$\text{Cover} : S^1 \rightarrow \text{Type}$

$\text{Cover}(\text{base}) := \mathbb{Z}$

$\text{Cover}_1(\text{loop}) :=$   
 $\text{ua}(\text{successor}) : \mathbb{Z} = \mathbb{Z}$

# Universal Cover



defined by circle  
recursion

$\text{Cover} : S^1 \rightarrow \text{Type}$

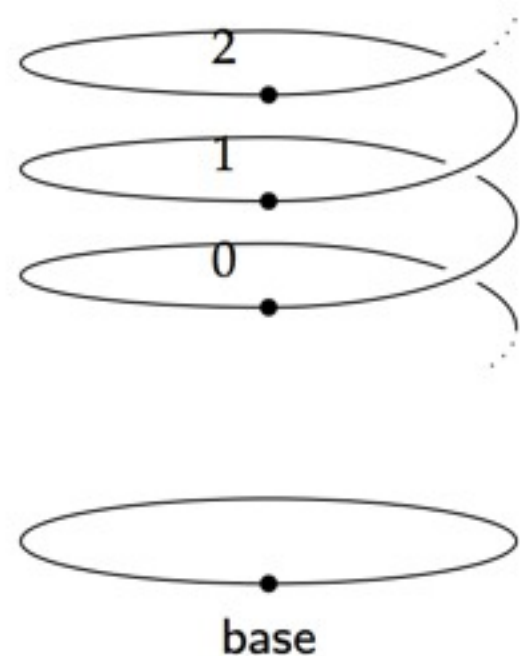
$\text{Cover}(\text{base}) := \mathbb{Z}$

$\text{Cover}_1(\text{loop}) :=$

$\text{ua}(\text{successor}) : \mathbb{Z} = \mathbb{Z}$

interpret loop as  
“add 1” bijection

# Universal Cover



$\mathbb{R}$   
 $\downarrow w$   
 $S^1$

defined by circle  
recursion

$\text{Cover} : S^1 \rightarrow \text{Type}$

$\text{Cover}(\text{base}) := \mathbb{Z}$

$\text{Cover}_1(\text{loop}) :=$

$\text{ua}(\text{successor}) : \mathbb{Z} = \mathbb{Z}$

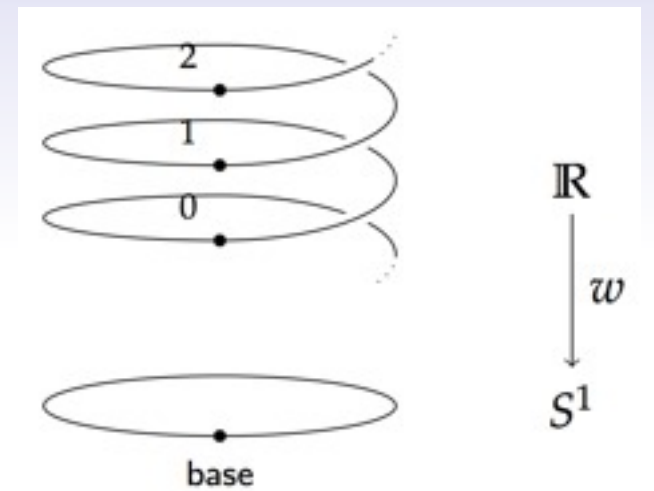
univalence

interpret loop as  
“add 1” bijection

# Winding number

$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

$$\text{wind}(p) = \text{transport}_{\text{cover}}(p, 0)$$



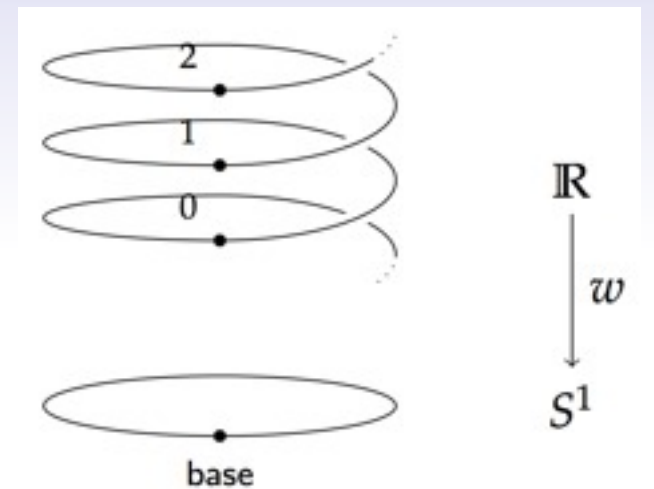
**lift  $p$  to cover,  
starting at 0**

# Winding number

$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

$$\text{wind}(p) = \text{transport}_{\text{cover}}(p, 0)$$

$$\text{wind}(\text{loop}^{-1} \circ \text{loop})$$



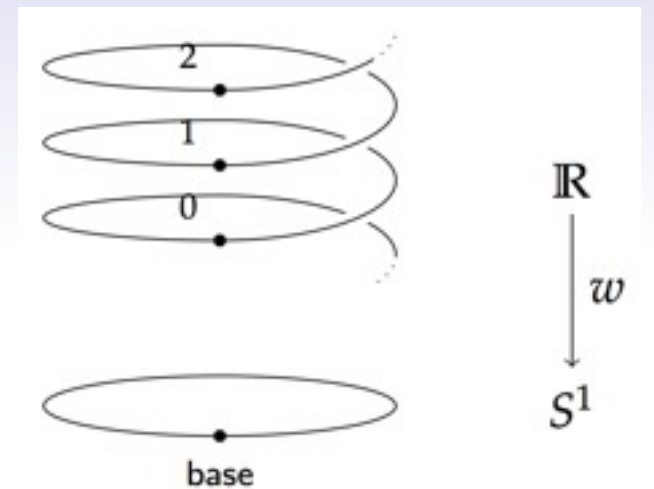
**lift p to cover,  
starting at 0**

# Winding number

$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

$$\text{wind}(p) = \text{transport}_{\text{cover}}(p, 0)$$

$$\begin{aligned} & \text{wind}(\text{loop}^{-1} \circ \text{loop}) \\ &= \text{transport}_{\text{cover}}(\text{loop}^{-1} \circ \text{loop}, 0) \end{aligned}$$



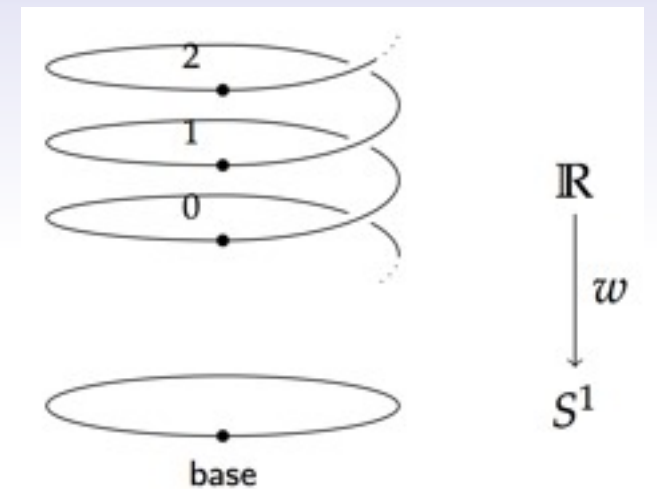
**lift p to cover,  
starting at 0**



# Winding number

$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

$$\text{wind}(p) = \text{transport}_{\text{Cover}}(p, 0)$$



**lift p to cover,  
starting at 0**

$$\text{wind}(\text{loop}^{-1} \circ \text{loop})$$

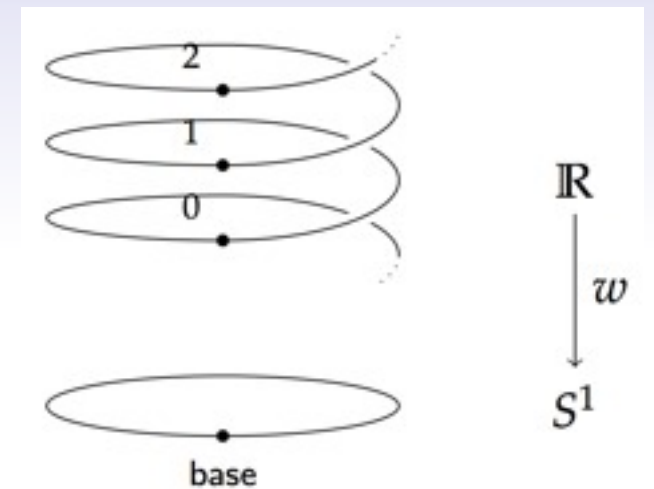
$$= \text{transport}_{\text{Cover}}(\text{loop}^{-1} \circ \text{loop}, 0)$$

$$= \text{transport}_{\text{Cover}}(\text{loop}^{-1}, \text{transport}_{\text{Cover}}(\text{loop}, 0))$$

# Winding number

$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

$$\text{wind}(p) = \text{transport}_{\text{Cover}}(p, 0)$$



**lift p to cover,  
starting at 0**

$$\text{wind}(\text{loop}^{-1} \circ \text{loop})$$

$$= \text{transport}_{\text{Cover}}(\text{loop}^{-1} \circ \text{loop}, 0)$$

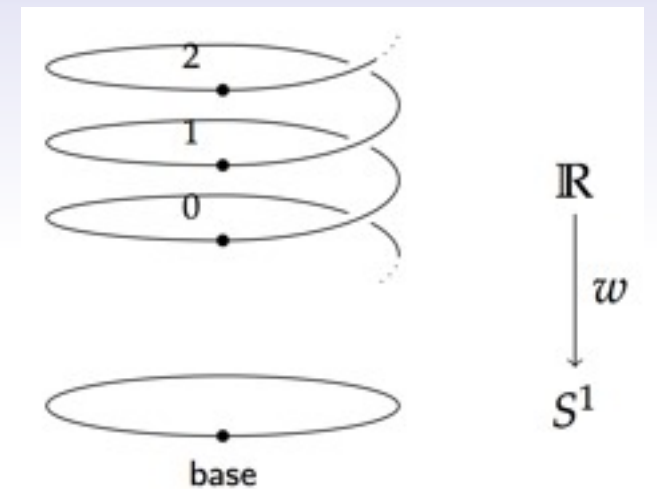
$$= \text{transport}_{\text{Cover}}(\text{loop}^{-1}, \text{transport}_{\text{Cover}}(\text{loop}, 0))$$

$$= \text{transport}_{\text{Cover}}(\text{loop}^{-1}, 1)$$

# Winding number

$$\text{wind} : \Omega(S^1) \rightarrow \mathbb{Z}$$

$$\text{wind}(p) = \text{transport}_{\text{Cover}}(p, 0)$$



**lift p to cover,  
starting at 0**

$$\text{wind}(\text{loop}^{-1} \circ \text{loop})$$

$$= \text{transport}_{\text{Cover}}(\text{loop}^{-1} \circ \text{loop}, 0)$$

$$= \text{transport}_{\text{Cover}}(\text{loop}^{-1}, \text{transport}_{\text{Cover}}(\text{loop}, 0))$$

$$= \text{transport}_{\text{Cover}}(\text{loop}^{-1}, 1)$$

$$= 0$$

# Fundamental group of the circle

## The HoTT book

## Agda

## 7.2.1.1 Encode/decode proof

By definition,  $\Omega(S^1)$  is base  $\Rightarrow$  base. If we attempt to prove that  $\Omega(S^1) = \mathbb{Z}$  by directly constructing an equivalence, we will get stuck, because type theory gives you little leverage for working with loops. Instead, we generalize the theorem statement to the path fibration, and analyze the whole fibration.

$$P(x : S^1) := \{ \text{base} \Rightarrow x \}$$

with one end-point free.

We show that  $P(x)$  is equal to another fibration, which gives a more explicit description of the paths—we call this other fibration “codes”, because its elements are data that act as codes for paths on the circle. In this case, the codes fibration is the universal cover of the circle.

**Definition 7.2.1** (Universal Cover of  $S^1$ ). Define  $\text{code}(x : S^1) : \mathbb{Z}$  by circle-recursion, with

$$\begin{aligned} \text{code}(\text{base}) &:= \mathbb{Z} \\ \text{code}(\text{loop}) &:= \text{us}(\text{succ}) \end{aligned}$$

where  $\text{succ}$  is the equivalence  $\mathbb{Z} \simeq \mathbb{Z}$  given by adding one, which by univalence determines a path from  $\mathbb{Z}$  to  $\mathbb{Z}$  in  $\mathcal{U}$ .

To define a function by circle recursion, we need to find a point and a loop in the target. In this case, the target is  $\mathbb{Z}$ , and the point we choose is  $\mathbb{Z}$ , corresponding to our expectation that the fiber of the universal cover should be the integers. The loop we choose is the successor/predecessor isomorphism on  $\mathbb{Z}$ , which corresponds to the fact that going around the loop in the base goes up one level on the helix. Univalence is necessary for this part of the proof, because we need a non-trivial equivalence on  $\mathbb{Z}$ .

From this definition, it is simple to calculate that transporting with code takes loop to the successor function, and  $\text{loop}^{-1}$  to the predecessor function:

**Lemma 7.2.2.**  $\text{transport}^{\text{code}}(\text{loop}, x) = x + 1$  and  $\text{transport}^{\text{code}}(\text{loop}^{-1}, x) = x - 1$

*Proof.* For the first, we calculate as follows:

$$\begin{aligned} & \text{transport}^{\text{code}}(\text{loop}, x) \\ &= \text{transport}^{\text{code}}(\text{code}(\text{loop}), x) \quad \text{associativity} \\ &= \text{transport}^{\text{code}}(\text{us}(\text{succ}), x) \quad \text{reduction for circle-recursion} \\ &= x + 1 \quad \text{reduction for us} \end{aligned}$$

The second follows from the first, because  $\text{transport}^p$  and  $\text{transport}^{p^{-1}}$  are always inverses, so  $\text{transport}^{\text{code}} \text{loop}^{-1} = \text{transport}^{\text{code}} \text{loop}^{-1}$  must be the inverse of the  $- + 1$ .

In the remainder of the proof, we will show that  $P$  and code are equivalent.

[DRAFT OF MARCH 19, 2013]

**7.2.1.1.1 Encoding** Next, we define a function  $\text{encode}$  that maps paths to codes:

**Definition 7.2.3.** Define  $\text{encode} : \prod (x : S^1), \rightarrow P(x) \rightarrow \text{code}(x)$  by

$$\text{encode } p := \text{transport}^{\text{code}}(p, 0)$$

(we leave the argument  $x$  implicit).

$\text{encode}$  is defined by lifting a path into the universal cover, which determines an equivalence, and then applying the resulting equivalence to 0. The interesting thing about this function is that it computes a concrete number from a loop on the circle, when this loop is represented using the abstract groupoidal framework of HoTT. To gain an intuition for how it does this, observe that by the above lemmas,  $\text{transport}^{\text{code}}(\text{loop}, x) = x + 1$  and  $\text{transport}^{\text{code}} \text{loop}^{-1} x = x - 1$ . Further, transport is functorial (chapter 2), so  $\text{transport}^{\text{code}} \text{loop} \circ \text{loop}$  is  $(\text{transport}^{\text{code}} \text{loop}) \circ (\text{transport}^{\text{code}} \text{loop})$ , etc. Thus, when  $p$  is a composition like

$$\text{loop} \circ \text{loop}^{-1} \circ \text{loop} \circ \dots$$

$\text{transport}^{\text{code}} p$  will compute a composition of functions like

$$\{ - + 1 \} \circ \{ - - 1 \} \circ \{ - + 1 \} \circ \dots$$

Applying this composition of functions to 0 will compute the winding number of the path—how many times it goes around the circle, with orientation marked by whether it is positive or negative, after inverses have been canceled. Thus, the computational behavior of  $\text{encode}$  follows from the reduction rules for higher-inductive types and univalence, and the action of transport on compositions and inverses.

Note that the instance  $\text{encode}' := \text{encode}_{\text{base}}$  has type  $\text{base} = \text{base} \rightarrow \mathbb{Z}$ , which will be one half of the equivalence between  $\text{base} = \text{base}$  and  $\mathbb{Z}$ .

**7.2.1.1.2 Decoding** Decoding an integer as a path is defined by recursion:

**Definition 7.2.4.** Define  $\text{loop}'' : \mathbb{Z} \rightarrow \text{base} = \text{base}$  by

$$\text{loop}'' = \begin{cases} \text{loop} \circ \text{loop} \circ \dots \circ \text{loop} \text{ (n times)} & \text{for positive } n \\ \text{loop}^{-1} \circ \text{loop}^{-1} \circ \dots \circ \text{loop}^{-1} \text{ (n times)} & \text{for negative } n \\ \text{refl} & \text{for } 0 \end{cases}$$

Since what we want overall is an equivalence between  $\text{base} = \text{base}$  and  $\mathbb{Z}$ , we might expect to be able to prove that  $\text{encode}'$  and  $\text{loop}''$  give an equivalence. The problem comes in trying to prove the “decode after encode” direction, where we would need to show that  $\text{loop}''(\text{encode } p) = p$  for all  $p$ . We would like to apply path induction, but path induction

[DRAFT OF MARCH 19, 2013]

does not apply to loops like  $a$  with both endpoints fixed! The way to solve this problem is to generalize the theorem to show that  $\text{loop}''(\text{encode } p) = p$  for all  $x : S^1$  and  $p : \text{base} \Rightarrow x$ . However, this does not make sense as is, because  $\text{loop}''$  is defined only for  $\text{base} = \text{base}$ , whereas here it is applied to a  $\text{base} \Rightarrow x$ . Thus, we generalize  $\text{loop}''$  as follows:

**Definition 7.2.5.** Define  $\text{decode} : \prod (x : S^1) \prod [\text{code}(x) \rightarrow P(x)]$ , by circle induction on  $x$ . It suffices to give a function  $\text{code}(\text{base}) \rightarrow P(\text{base})$ , for which we use  $\text{loop}''$ , and to show that  $\text{loop}''$  respects the loop.

*Proof.* To show that  $\text{loop}''$  respects the loop, it suffices to give a path from  $\text{loop}''$  to itself that lies over  $\text{loop}$ . Formally, this means a path from  $\text{transport}^{(\text{code} \circ \text{loop})}(\text{loop}, \text{loop}'')$  to  $\text{loop}''$ . We define such a path as follows:

$$\begin{aligned} & \text{transport}^{(\text{code} \circ \text{loop})}(\text{loop}, \text{loop}'') \\ &= \text{transport}^{\text{loop} \circ \text{loop}''} \circ \text{transport}^{\text{code}} \text{loop}^{-1} \\ &= \{ - + \text{loop} \} \circ \{ \text{loop}'' \} \circ \text{transport}^{\text{code}} \text{loop}^{-1} \\ &= \{ - + \text{loop} \} \circ \{ \text{loop}'' \} \circ \{ - - 1 \} \\ &= \{ n \mapsto \text{loop}^{n-1} \circ \text{loop} \} \end{aligned}$$

From line 1 to line 2, we apply the definition of transport when the outer connective of the fibration is  $\circ$ , which reduces the transport to pre- and post-composition with transport at the domain and range types. From line 2 to line 3, we apply the definition of transport when the type family is  $\text{base} \Rightarrow x$ , which is post-composition of paths. From line 3 to line 4, we use the action of code on  $\text{loop}^{-1}$  defined in Lemma 7.2.2. From line 4 to line 5, we simply reduce the function composition. Thus, it suffices to show that for all  $n$ ,  $\text{loop}^{n-1} \circ \text{loop} = \text{loop}''$ , which is an easy induction, using the groupoid laws.  $\square$

## 7.2.1.1.3 Decoding after encoding

**Lemma 7.2.6.** For all  $p$  for all  $x : S^1$  and  $p : \text{base} \Rightarrow x$ ,  $\text{decode}_x(\text{encode}_x(p)) = p$ .

*Proof.* By path induction, it suffices to show that  $\text{decode}_{\text{base}}(\text{encode}_{\text{base}}(\text{refl}_{\text{base}})) = \text{refl}_{\text{base}}$ . But  $\text{encode}_{\text{base}}(\text{refl}_{\text{base}}) = \text{transport}^{\text{code}}(\text{refl}_{\text{base}}, 0) = 0$ , and  $\text{decode}_{\text{base}}(0) = \text{loop}'' = \text{refl}_{\text{base}}$ .  $\square$

## 7.2.1.1.4 Encoding after decoding

**Lemma 7.2.7.** For all  $p$  for all  $x : S^1$  and  $c : \text{code}(x)$ ,  $\text{encode}_x(\text{decode}_x(c)) = c$ .

*Proof.* The proof is by circle induction. It suffices to show the case for base, because the case for loop is a path between paths in  $\mathbb{Z}$ , which can be given by appealing to the fact that  $\mathbb{Z}$  is a set.

Thus, it suffices to show, for all  $n : \mathbb{Z}$ , that

$$\text{encode}(\text{loop}'' n) = n$$

The proof is by induction, with cases for 0,  $-1$ ,  $-1x + 1$ , and  $n + 1$ .

- In the case for 0, the result is true by definition.
- In the case for 1,  $\text{encode}(\text{loop}'')$  reduces to  $\text{transport}^{\text{code}}(\text{loop}, 0)$ , which by Lemma 7.2.2 is  $0 + 1 = 1$ .
- In the case for  $n + 1$ ,

$$\begin{aligned} & \text{encode}(\text{loop}''(n+1)) \\ &= \text{encode}(\text{loop}'' \circ \text{loop}) \\ &= \text{transport}^{\text{code}}(\text{loop}'' \circ \text{loop}, 0) \\ &= \text{transport}^{\text{code}}(\text{loop}, \text{transport}^{\text{code}}(\text{loop}'', 0)) \quad \text{by functoriality} \\ &= (\text{transport}^{\text{code}}(\text{loop}'', 0)) + 1 \quad \text{by Lemma 7.2.2} \\ &= n + 1 \quad \text{by the IH} \end{aligned}$$

- The cases for negatives are analogous.  $\square$

## 7.2.1.1.5 Tying it all together

**Theorem 7.2.8.** There is a family of equivalences  $\prod (x : S^1) \prod [P(x) \simeq \text{code}(x)]$ .

*Proof.* The maps  $\text{encode}$  and  $\text{decode}$  are mutually inverse by Lemmas 7.2.6 and 7.2.7, and this can be improved to an equivalence.  $\square$

Instantiating at base gives

**Corollary 7.2.9.**  $(\text{base} = \text{base}) \simeq \mathbb{Z}$

A simple induction shows that this equivalence takes addition to composition, so  $\Omega(S^1) = \mathbb{Z}$  as groups.

**Corollary 7.2.10.**  $\pi_k(S^1) = \mathbb{Z}$  if  $k = 1$  and 0 otherwise.

*Proof.* For  $k = 1$ , we sketched the proof from Corollary 7.2.9 above. For  $k > 1$ ,  $\|\Omega^{k+1}(S^1)\|_0 = \|\Omega^k(\Omega(S^1))\|_0 = \|\Omega^k(\mathbb{Z})\|_0$ , which is 1 because  $\mathbb{Z}$  is a set and  $\pi_n$  of a set is trivial (PIDM lemmas to cite!).  $\square$

[DRAFT OF MARCH 19, 2013]

Cover :  $S^1 \rightarrow \text{Type}$   
Cover x =  $S^1 \rightarrow \text{rec } \text{Int} \text{ (us succEquiv) } x$

transport-Cover-loop : Path (transport Cover loop) succ  
transport-Cover-loop =  
transport Cover loop  
=< transport-ap-assoc Cover loop >  
transport (λ x → x) (ap Cover loop)  
=< ap (transport (λ x → x)) (ap Cover loop) >  
(gloop/rec Int (us succEquiv)) >  
transport (λ x → x) (us succEquiv)  
=< typeβ \_ >  
succ \*

transport-Cover-ll-loop : Path (transport Cover (l loop)) pred  
transport-Cover-ll-loop =  
transport Cover (l loop)  
=< transport-ap-assoc Cover (l loop) >  
transport (λ x → x) (ap Cover (l loop))  
=< ap (transport (λ x → x)) (ap Cover (l loop)) >  
transport (λ x → x) (l (ap Cover loop))  
=< ap (λ y → transport (λ x → x) (l y))  
(gloop/rec Int (us succEquiv)) >  
transport (λ x → x) (l (us succEquiv))  
=< ap (transport (λ x → x)) (l (us succEquiv)) >  
transport (λ x → x) (us (l equiv succEquiv))  
=< typeβ \_ >  
pred \*

encode :  $\{x : S^1\} \rightarrow \text{Path base } x \rightarrow \text{Cover } x$   
encode a = transport Cover a Zero

encode' : Path base base → Int  
encode' a = encode (base) a

loopA : Int → Path base base  
loopA Zero = id  
loopA (Pos One) = loop  
loopA (Pos (S n)) = loop · loopA (Pos n)  
loopA (Neg One) = l loop  
loopA (Neg (S n)) = l loop · loopA (Neg n)

loopA-preserves-pred  
: (n : Int) → Path (loopA (pred n)) (l loop · loopA n)  
loopA-preserves-pred (Pos One) = l (l-inv-1 loop)  
loopA-preserves-pred (Pos (S y)) =  
l (l-assoc (l loop) loop (loopA (Pos y)))  
· l (ap (λ x → x · loopA (Pos y)) (l-inv-1 loop))  
· l (l-unit-1 (loopA (Pos y)))  
loopA-preserves-pred Zero = id  
loopA-preserves-pred (Neg One) = id  
loopA-preserves-pred (Neg (S y)) = id

decode :  $\{x : S^1\} \rightarrow \text{Cover } x \rightarrow \text{Path base } x$   
decode (x) =

$S^1$ -induction

(λ x' → Cover x' → Path base x')

loopA

loopA-respects-loop

x where

abstract -- prevent Agda from normalizing

loopA-respects-loop : transport (λ x' → Cover x' → Path base x') loop loopA = (λ n → loopA n)  
loopA-respects-loop =  
(transport (λ x' → Cover x' → Path base x') loop loopA  
=< transport-- Cover (Path base) loop loopA >  
transport (λ x' → Path base x') loop  
· loopA  
· transport Cover (l loop)  
=< λ y → transport-path-right loop (loopA (transport Cover (l loop) y))) >  
(λ p → loop · p)  
· loopA  
· transport Cover (l loop)  
=< λ y → ap (λ x' → loop · loopA x') (ap transport-Cover-ll-loop)) >  
(λ p → loop · p)  
· loopA  
· pred  
=< id >  
(λ n → loop · (loopA (pred n)))  
=< λ y → move-left-l \_ loop (loopA y) (loopA-preserves-pred y)) >  
(λ n → loopA n)  
\*)

abstract -- prevent Agda from normalizing  
encode-loopA : (n : Int) → Path (encode (loopA n)) n  
encode-loopA Zero = id  
encode-loopA (Pos One) = ap transport-Cover-loop  
encode-loopA (Pos (S n)) =  
encode (loopA (Pos (S n)))  
=< id >  
transport Cover (loop · loopA (Pos n)) Zero  
=< ap (transport-- Cover loop (loopA (Pos n))) >  
transport Cover loop  
(transport Cover (loopA (Pos n)) Zero)  
=< ap transport-Cover-loop >  
succ (transport Cover (loopA (Pos n)) Zero)  
=< id >  
succ (encode (loopA (Pos n)))  
=< ap succ (encode-loopA (Pos n)) >  
succ (Pos n) \*

encode-decode :  $\{x : S^1\} \rightarrow \{c : \text{Cover } x\}$   
→ Path (encode (decode (x) c)) c  
encode-decode (x) =  $S^1$ -induction  
(λ (x : S^1) → {c : Cover x}  
→ Path (encode (x) (decode (x) c)) c)  
encode-loopA (λ x' → fst (use-level (use-level HSet-Int \_ \_) \_ \_)) x

decode-encode :  $\{x : S^1\} \rightarrow \{a : \text{Path base } x\}$   
→ Path (decode (encode a)) a  
decode-encode (x) =  
path-induction  
(λ (x' : S^1) (x' : Path base x')  
→ Path (decode (encode a')) a')  
id =

Ω[S<sup>1</sup>]-Equiv-Int : Equiv (Path base base) Int  
Ω[S<sup>1</sup>]-Equiv-Int =  
improve (hequiv encode decode decode-encode encode-loopA)

Ω[S<sup>1</sup>]-is-Int : (Path base base) = Int  
Ω[S<sup>1</sup>]-is-Int = ua Ω[S<sup>1</sup>]-Equiv-Int

n[S<sup>1</sup>]-is-Int : a One S<sup>1</sup> base = Int  
n[S<sup>1</sup>]-is-Int = UnTrunc.path \_ \_ HSet-Int · ap (Trunc (λ l 0)) Ω[S<sup>1</sup>]-is-Int



# $\pi_n(S^n)$ in HoTT

$k^{\text{th}}$  homotopy group

n-dimensional sphere

	$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$	$\pi_5$	$\pi_6$	$\pi_7$	$\pi_8$	$\pi_9$	$\pi_{10}$	$\pi_{11}$	$\pi_{12}$	$\pi_{13}$	$\pi_{14}$	$\pi_{15}$
$S^0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$S^1$	$\mathbb{Z}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$S^2$	0	$\mathbb{Z}$	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_3$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{12} \times \mathbb{Z}_2$	$\mathbb{Z}_{84} \times \mathbb{Z}_2^2$	$\mathbb{Z}_2^2$
$S^3$	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_3$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{12} \times \mathbb{Z}_2$	$\mathbb{Z}_{84} \times \mathbb{Z}_2^2$	$\mathbb{Z}_2^2$
$S^4$	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z} \times \mathbb{Z}_{12}$	$\mathbb{Z}_2^2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{24} \times \mathbb{Z}_3$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^3$	$\mathbb{Z}_{120} \times \mathbb{Z}_{12} \times \mathbb{Z}_2$	$\mathbb{Z}_{84} \times \mathbb{Z}_2^5$
$S^5$	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{30}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^3$	$\mathbb{Z}_{72} \times \mathbb{Z}_2$
$S^6$	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_{60}$	$\mathbb{Z}_{24} \times \mathbb{Z}_2$	$\mathbb{Z}_2^3$
$S^7$	0	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_{120}$	$\mathbb{Z}_2^3$
$S^8$	0	0	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	0	$\mathbb{Z}_2$	$\mathbb{Z} \times \mathbb{Z}_{120}$

[image from wikipedia]

# $\pi_n(S^n)$ in HoTT

$k^{\text{th}}$  homotopy group

n-dimensional sphere

	$\pi_1$	$\pi_2$	$\pi_3$	$\pi_4$	$\pi_5$	$\pi_6$	$\pi_7$	$\pi_8$	$\pi_9$	$\pi_{10}$	$\pi_{11}$	$\pi_{12}$	$\pi_{13}$	$\pi_{14}$	$\pi_{15}$
$S^0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$S^1$	$\mathbb{Z}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$S^2$	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_3$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{12} \times \mathbb{Z}_2$	$\mathbb{Z}_{84} \times \mathbb{Z}_2^2$	$\mathbb{Z}_2^2$
$S^3$	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{12}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_3$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{12} \times \mathbb{Z}_2$	$\mathbb{Z}_{84} \times \mathbb{Z}_2^2$	$\mathbb{Z}_2^2$
$S^4$	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z} \times \mathbb{Z}_{12}$	$\mathbb{Z}_2^2$	$\mathbb{Z}_2^2$	$\mathbb{Z}_{24} \times \mathbb{Z}_3$	$\mathbb{Z}_{15}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^3$	$\mathbb{Z}_{120} \times \mathbb{Z}_{12} \times \mathbb{Z}_2$	$\mathbb{Z}_{84} \times \mathbb{Z}_2^5$
$S^5$	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{30}$	$\mathbb{Z}_2$	$\mathbb{Z}_2^3$	$\mathbb{Z}_{72} \times \mathbb{Z}_2$
$S^6$	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_{60}$	$\mathbb{Z}_{24} \times \mathbb{Z}_2$	$\mathbb{Z}_2^3$
$S^7$	0	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	0	$\mathbb{Z}_2$	$\mathbb{Z}_{120}$	$\mathbb{Z}_2^3$
$S^8$	0	0	0	0	0	0	0	$\mathbb{Z}$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}_{24}$	0	0	$\mathbb{Z}_2$	$\mathbb{Z} \times \mathbb{Z}_{120}$

[image from wikipedia]

$$\pi_n(S^n) = \mathbb{Z} \text{ for } n \geq 1$$

**Proof:** Induction on  $n$

\* Base case:  $\pi_1(S^1) = \mathbb{Z}$

\* Inductive step:  $\pi_{n+1}(S^{n+1}) = \pi_n(S^n)$

$$\pi_n(S^n) = \mathbb{Z} \text{ for } n \geq 1$$

**Proof:** Induction on  $n$

\* Base case:  $\pi_1(S^1) = \mathbb{Z}$

\* Inductive step:  $\pi_{n+1}(S^{n+1}) = \pi_n(S^n)$

Key lemma:  $|S^n|_n = |\Omega(S^{n+1})|_n$



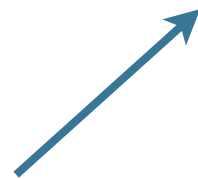
$$\pi_n(S^n) = \mathbb{Z} \text{ for } n \geq 1$$

**Proof:** Induction on  $n$

\* Base case:  $\pi_1(S^1) = \mathbb{Z}$

\* Inductive step:  $\pi_{n+1}(S^{n+1}) = \pi_n(S^n)$

Key lemma:  $|S^n|_n = |\Omega(S^{n+1})|_n$



**n-truncation:**

**best approximation of a type such  
that all  $(n+1)$ -paths are equal**

$$\pi_n(S^n) = \mathbb{Z} \text{ for } n \geq 1$$

**Proof:** Induction on  $n$

\* Base case:  $\pi_1(S^1) = \mathbb{Z}$

\* Inductive step:  $\pi_{n+1}(S^{n+1}) = \pi_n(S^n)$

Key lemma:  $|S^n|_n = |\Omega(S^{n+1})|_n$

**n-truncation:**  
best approximation of a type such  
that all  $(n+1)$ -paths are equal

**higher inductive type  
generated by**  
 $\text{base}_n : S^n$   
 $\text{loop}_n : \Omega^n(S^n)$

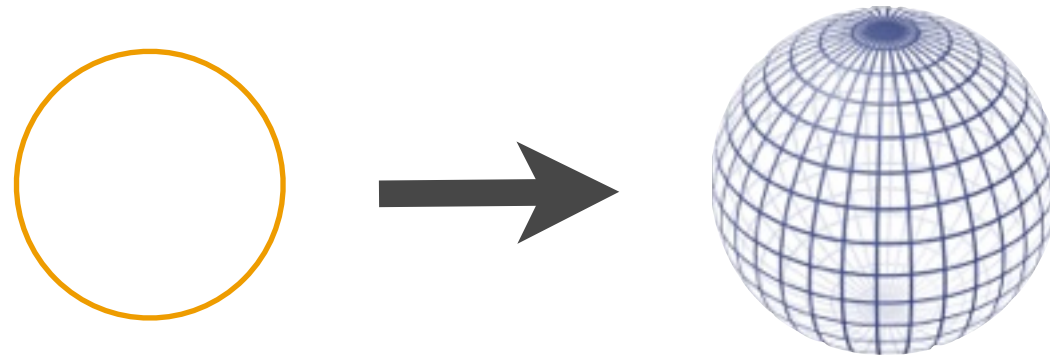
$$|S^n|_n = |\Omega(S^{n+1})|_n$$

$n$ -truncation of  $S^n$  is the type of “codes” for loops on  $S^{n+1}$

$$|S^n|_n = |\Omega(S^{n+1})|_n$$

$n$ -truncation of  $S^n$  is the type of “codes” for loops on  $S^{n+1}$

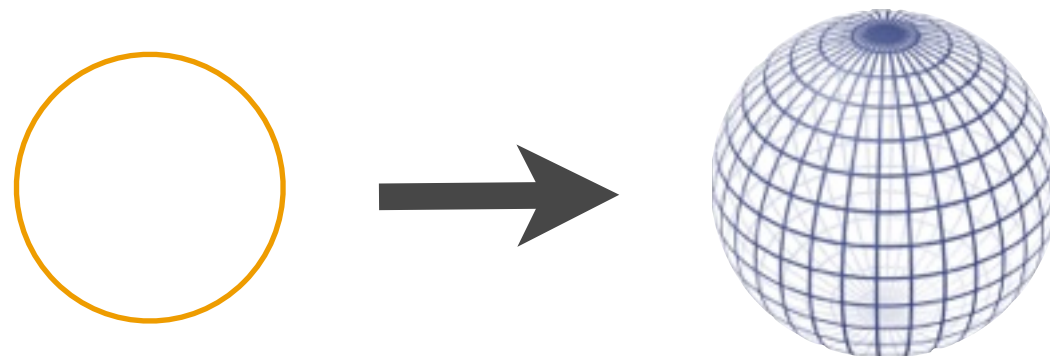
- \* Decode: promote  $n$ -dimensional loop on  $S^n$  to  $n+1$ -dimensional loop on  $S^{n+1}$



$$|S^n|_n = |\Omega(S^{n+1})|_n$$

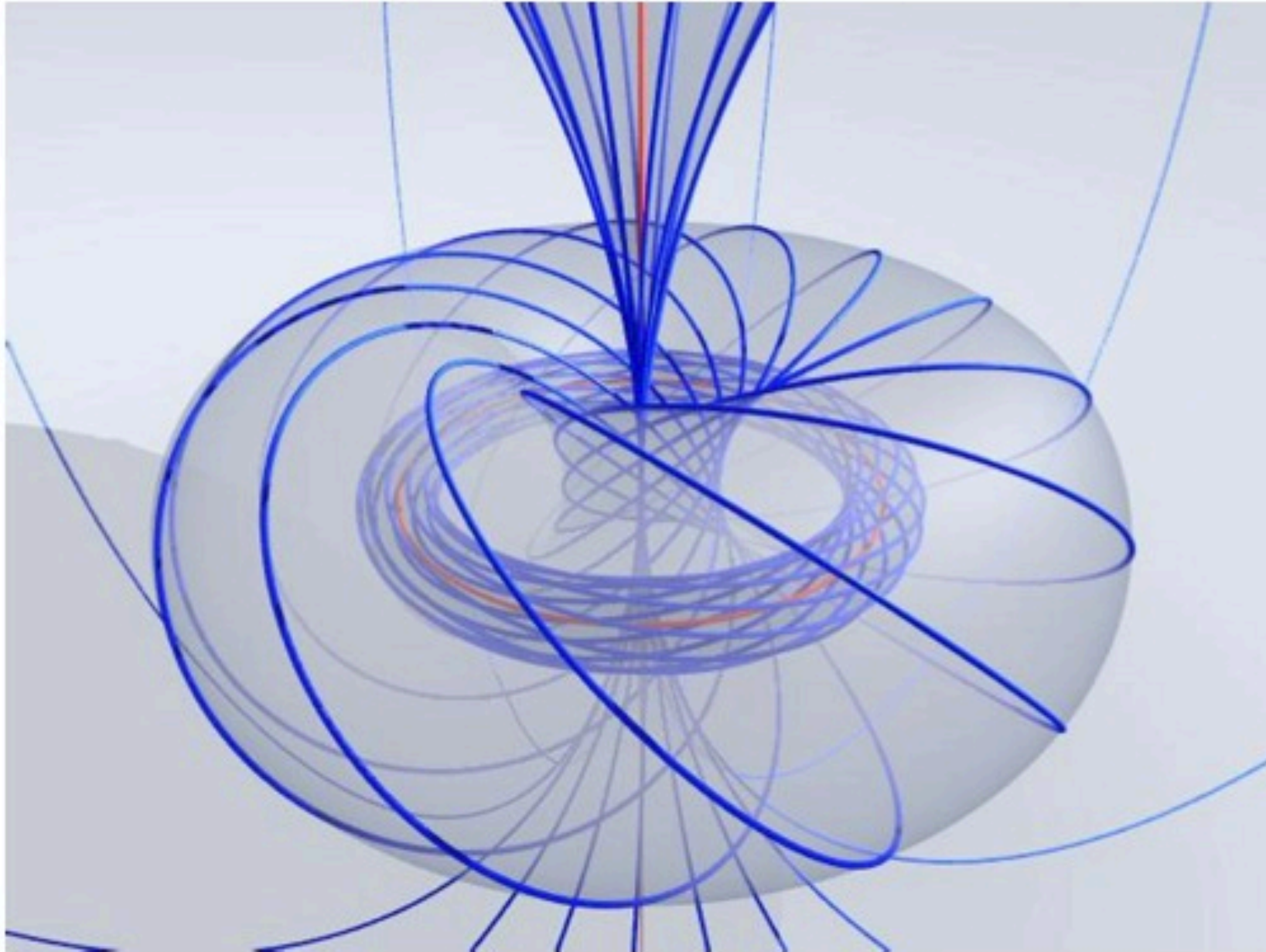
$n$ -truncation of  $S^n$  is the type of “codes” for loops on  $S^{n+1}$

- \* Decode: promote  $n$ -dimensional loop on  $S^n$  to  $n+1$ -dimensional loop on  $S^{n+1}$



- \* Encode: define fibration  $\text{Code}(x : S^{n+1})$  with  
 $\text{Code}(\text{base}_{n+1}) := |S^n|_n$   
 $\text{Code}(\text{loop}_{n+1}) := \text{equivalence } |S^n|_n \simeq |S^n|_n$   
“rotating by  $\text{loop}_n$ ”

# $\pi_2(S^2)$ : Hopf fibration



©Benoît R. Kloeckner CC-BY-NC

# Synthetic homotopy theory

- \* Gap between informal and formal proofs is small
- \* Proofs are constructive\*: can run them
- \* Results apply in a variety of settings,  
from simplicial sets (hence topological spaces)  
to Quillen model categories and  $\infty$ -topoi\*
- \* New type-theoretic proofs/methods

\*work in progress



# Homotopy Type Theory

*Univalent Foundations of Mathematics*

$\Sigma \Pi \Sigma \Pi \times \Pi \Sigma \Pi \Sigma \Pi \Sigma \Pi \lambda \Pi \Sigma \simeq$   
 $\Sigma \Pi \simeq \lambda \simeq \times \simeq \Sigma \simeq \times \simeq \simeq \times \simeq \Sigma \simeq \lambda \Pi \Sigma \Pi \Sigma \simeq$   
 $\Pi \simeq \lambda \times \Sigma \Pi \Sigma \Pi \lambda \Pi \lambda \Pi \Sigma \Pi \Sigma \Pi \lambda \Pi \times \Sigma \simeq \times \lambda \Pi \lambda \Pi$   
 $\lambda \times \Sigma \simeq \times \simeq \simeq \Sigma \simeq \lambda \simeq \lambda \simeq \Sigma \Pi \lambda \Pi \Sigma \simeq \Sigma \simeq \Sigma$   
 $\Pi \Pi \times \Sigma \Pi \Sigma \Pi \times \Pi \times \Sigma \Pi \Sigma \Pi \times \Pi \simeq \times \Sigma \times \Pi \lambda \Pi \times \lambda$   
 $\simeq \Sigma \simeq \lambda \simeq \times \lambda \simeq \lambda \simeq \lambda \Sigma \lambda \Pi \simeq \Pi \Sigma \simeq \Sigma \simeq \Pi$   
 $\Pi \Pi \Sigma \Pi \Sigma \Pi \Sigma \Pi \Sigma \Pi \simeq \Pi \simeq \lambda \Sigma \lambda \Pi \lambda \Sigma$   
 $\Pi \simeq \times \lambda \simeq \lambda \simeq \simeq \Sigma \Pi \simeq \Sigma \simeq \lambda \Sigma \Pi \simeq$   
 $\Pi \Sigma \Pi \Sigma \Sigma \Pi \times \Pi \Sigma \Pi \Sigma \Pi \simeq \Pi \lambda \Sigma \Pi \times \Pi \simeq$   
 $\simeq \times \simeq \times \simeq \Pi \lambda \simeq \simeq \lambda \Pi \simeq \Sigma \simeq \Sigma \times \Sigma \Pi$   
 $\Sigma \Pi \Sigma \lambda \Sigma \Pi \Sigma \Pi \Sigma \Pi \Sigma \Pi \Sigma \Pi \times \Sigma \times \Pi \lambda \Pi \simeq \Pi \simeq \lambda$   
 $\simeq \times \simeq \Pi \simeq \lambda \simeq \simeq \lambda \simeq \lambda \simeq \lambda \Pi \simeq \Sigma \Sigma \lambda \Sigma \Pi$   
 $\Sigma \Pi \Sigma \times \Sigma \times \Sigma \Pi \lambda \Pi \Sigma \Pi \Sigma \Pi \Sigma \Pi \Sigma \times \Sigma \Pi \simeq \Pi \simeq \Pi \simeq \lambda \Sigma$   
 $\simeq \times \simeq \Pi \simeq \Pi \times \Sigma \simeq \times \simeq \times \simeq \times \simeq \lambda \Pi \simeq \times \Sigma \lambda \Sigma \lambda \Sigma \times \Pi$   
 $\Pi \Sigma \lambda \Sigma \lambda \Sigma \lambda \Pi \lambda \Pi \Sigma \Pi \Sigma \Pi \Sigma \Pi \Sigma \times \Sigma \Pi \simeq \Pi \simeq \Pi \simeq \lambda \lambda$   
 $\times \simeq \Pi \simeq \Pi \times \Sigma \times \Sigma \lambda \Sigma \times \Sigma \Pi \Sigma$   
 $\Pi \Sigma \times \Sigma \lambda \Sigma \Pi \simeq \Pi \simeq \Pi \times \simeq$   
 $\simeq \lambda \Pi \simeq \Pi \simeq \Pi \Sigma \lambda \Sigma \lambda \times \Sigma \Pi \Sigma \Pi$   
 $\Pi \Sigma \Sigma \times \Sigma \simeq \times \Pi \Sigma \simeq \Sigma \Sigma \Pi \Sigma \lambda \Pi \simeq \Pi \simeq \Pi \simeq \times \simeq \lambda$   
 $\Pi \times \Pi \simeq \Sigma \times \Sigma \times \lambda \Pi \Sigma \Sigma \Pi \Sigma \Pi \Sigma \Pi \Sigma$   
 $\Pi \Sigma \Pi \lambda \Pi \lambda \Pi \times \Pi \simeq \simeq \Sigma \lambda \Pi \Sigma$   
 $\Sigma \simeq \Sigma \simeq \Pi \lambda \Pi \Sigma \simeq \Sigma \simeq \Sigma \lambda \Sigma \Pi \Sigma \Pi \times \Pi \simeq \Pi \times$

THE UNIVALENT FOUNDATIONS PROGRAM  
INSTITUTE FOR ADVANCED STUDY

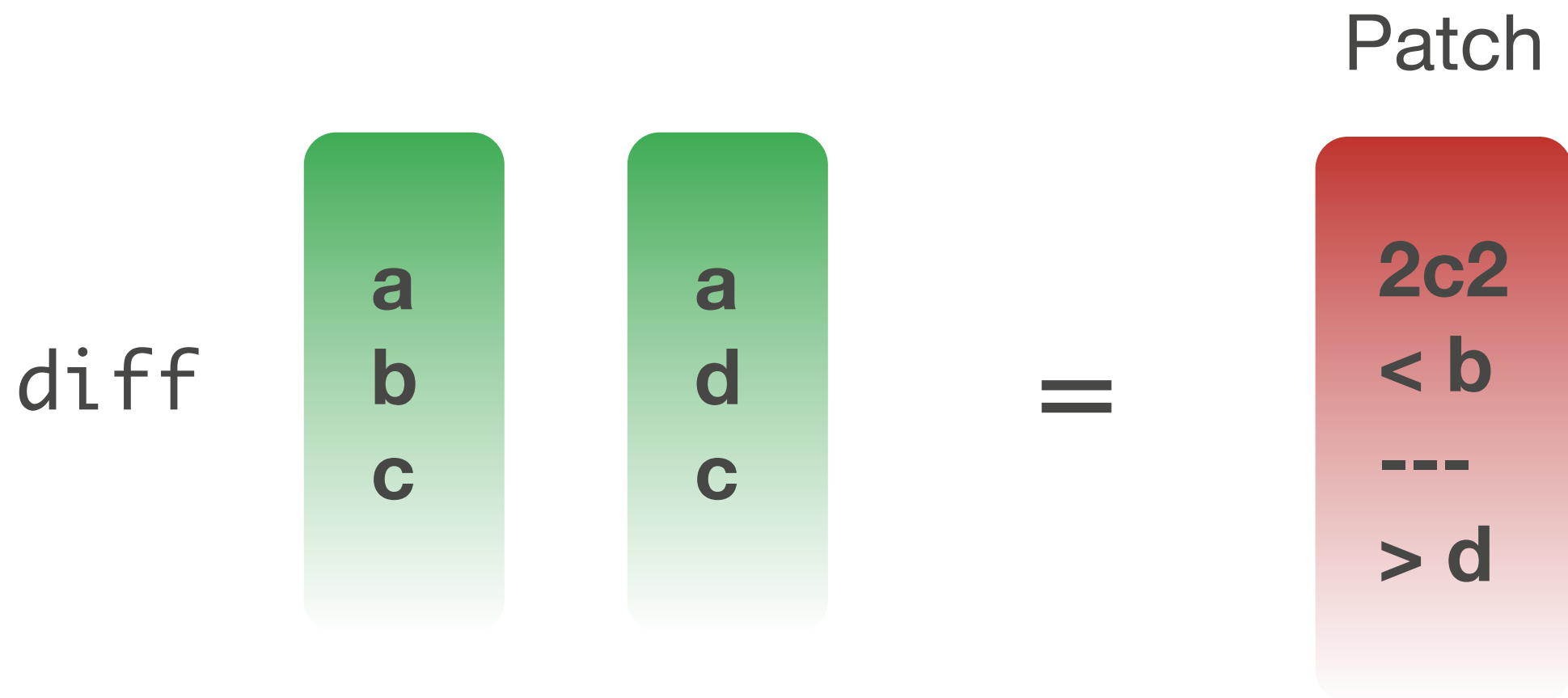


# Outline

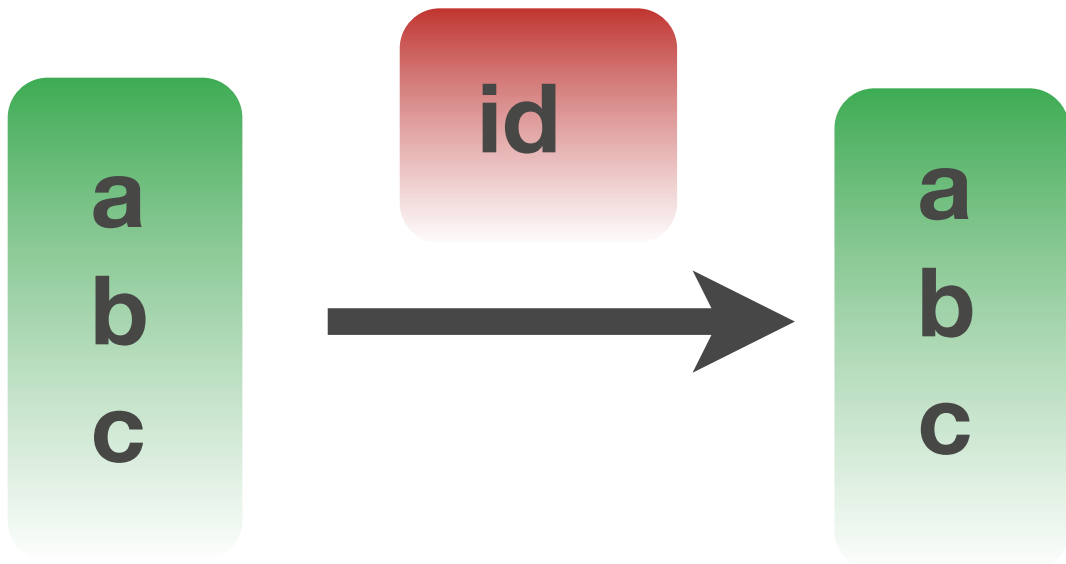
1.Certified homotopy theory

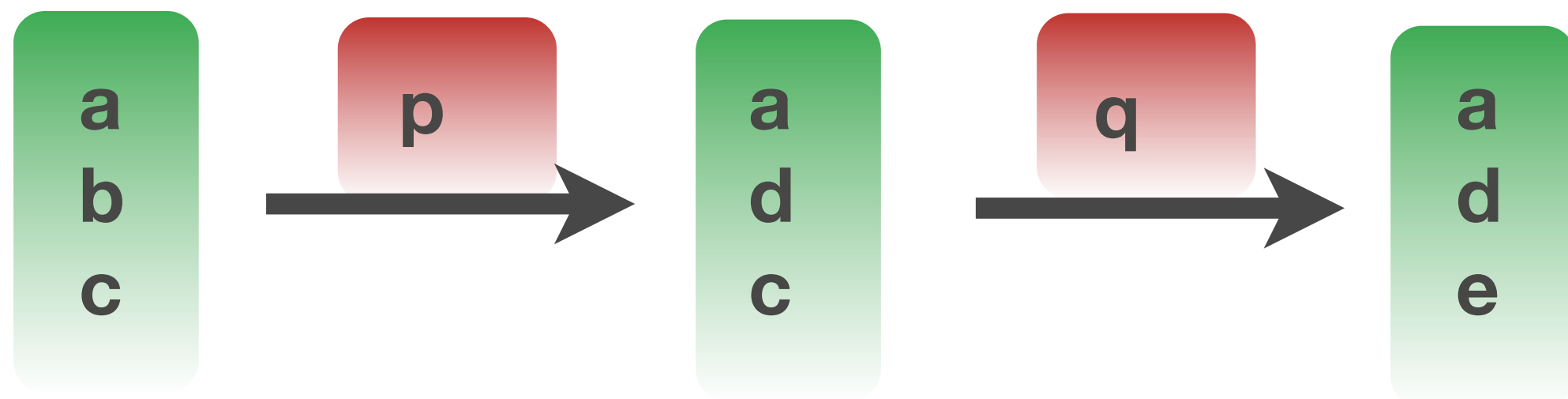
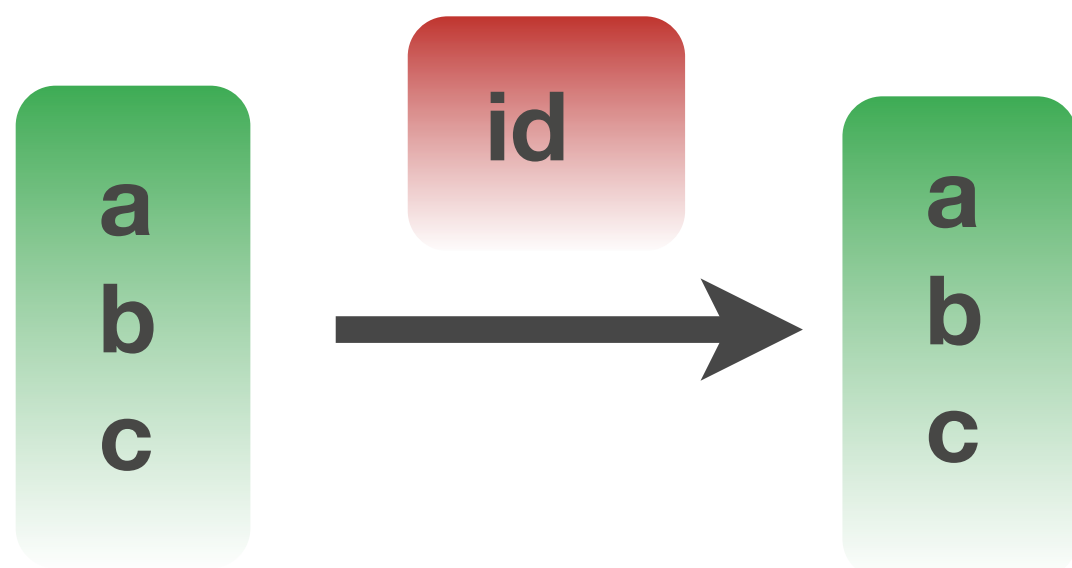
**2.Certified software**

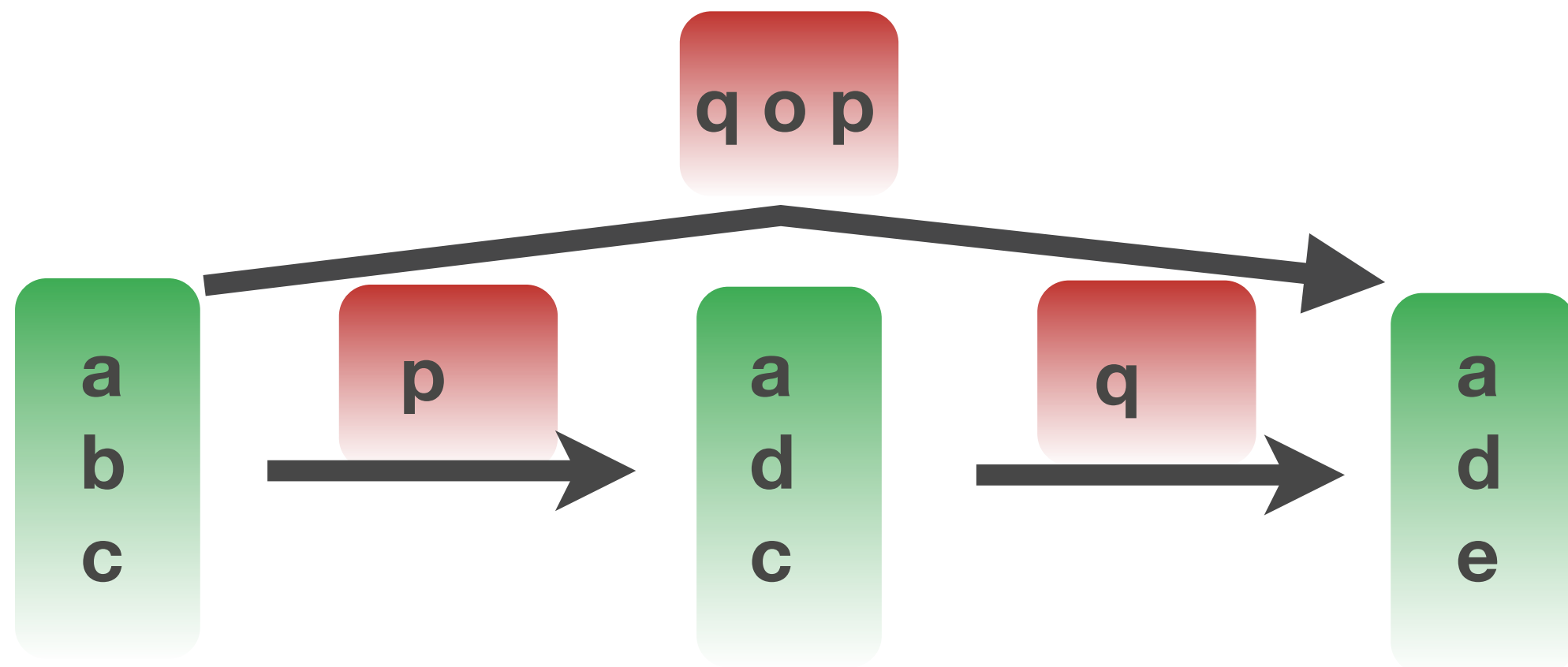
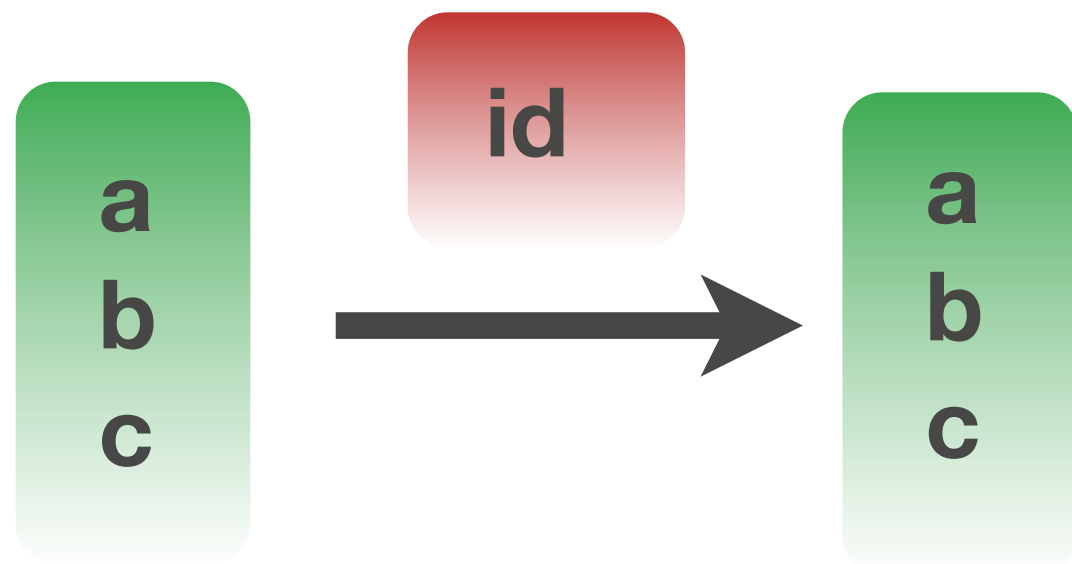
# Patches

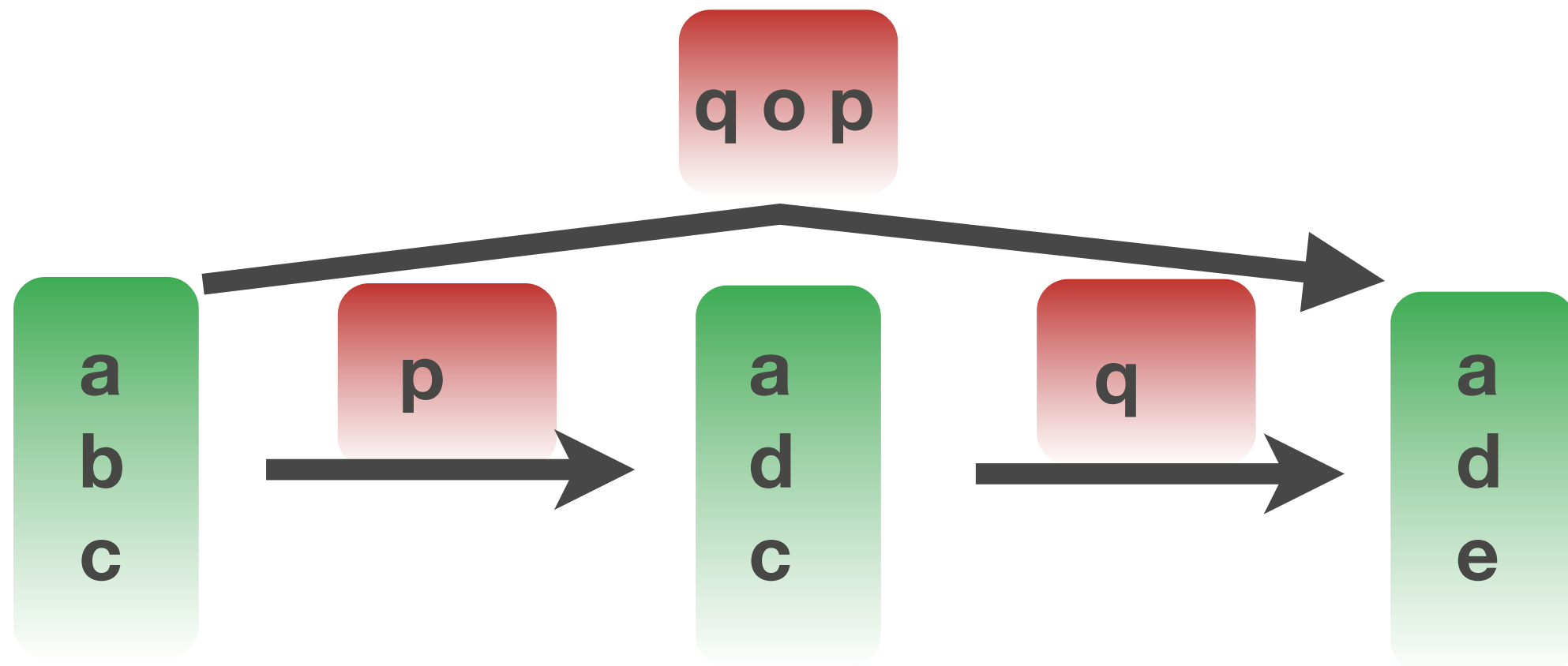
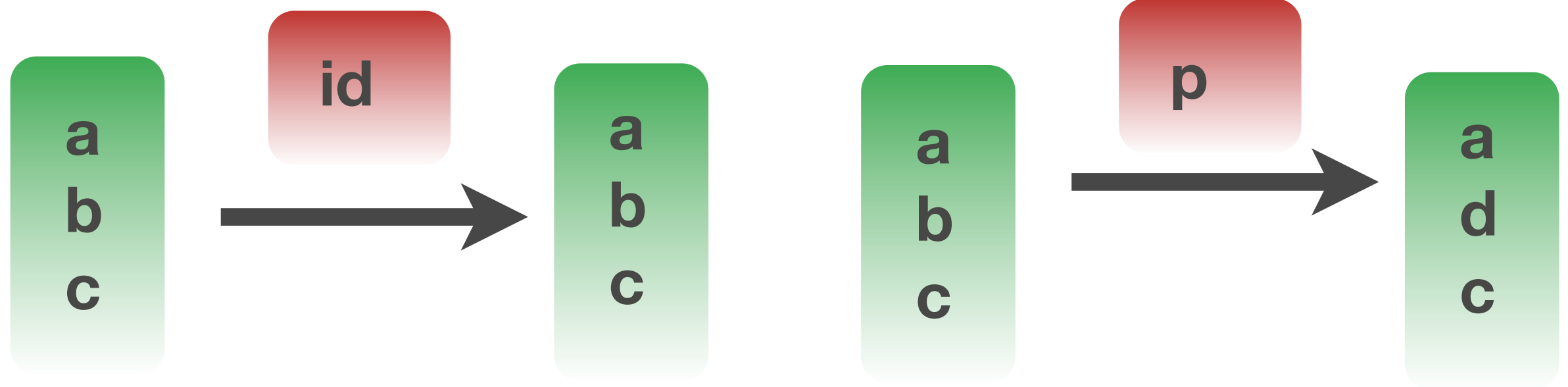


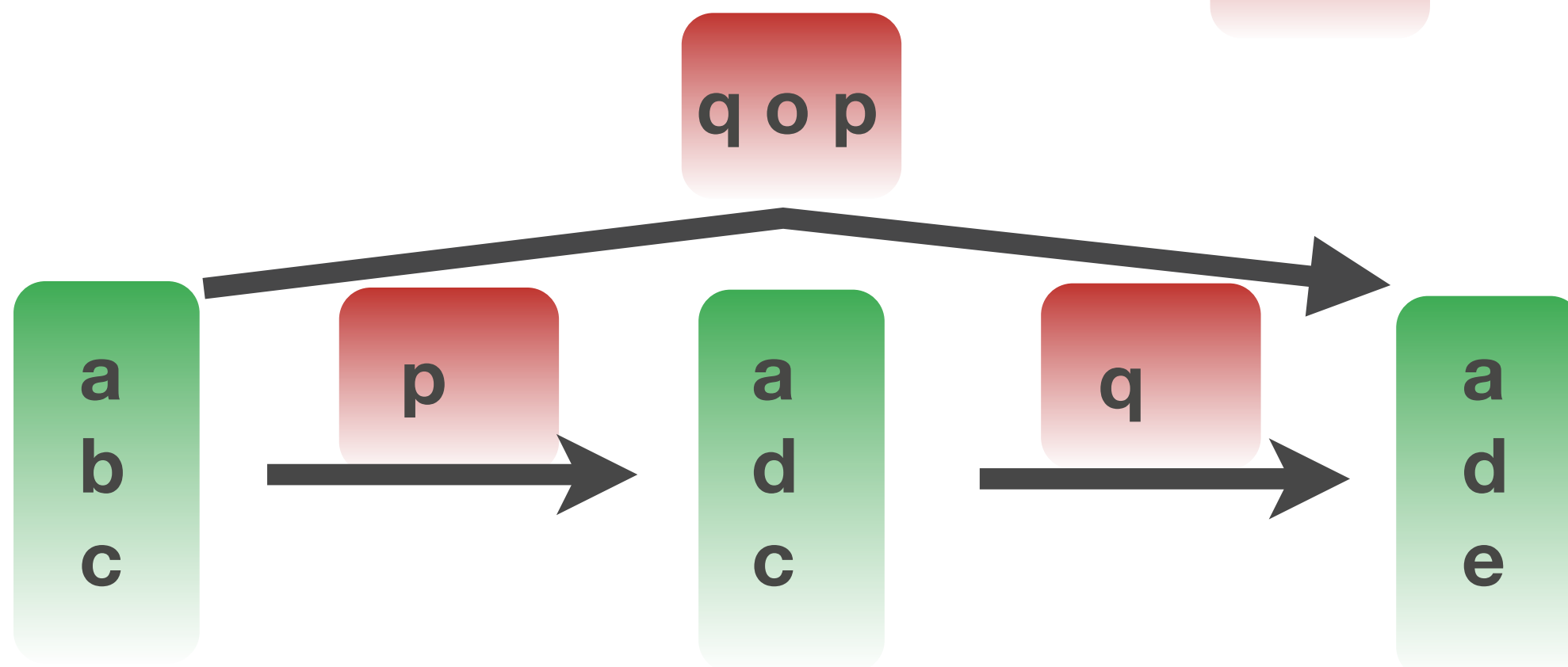
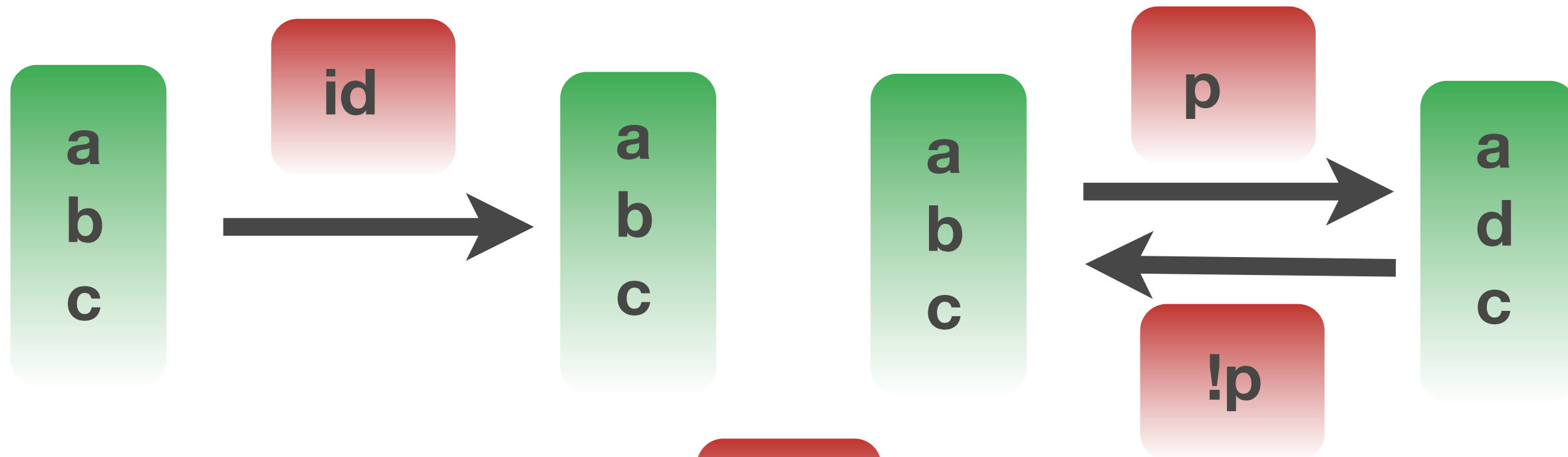
- \* Version control
- \* Collaborative editing

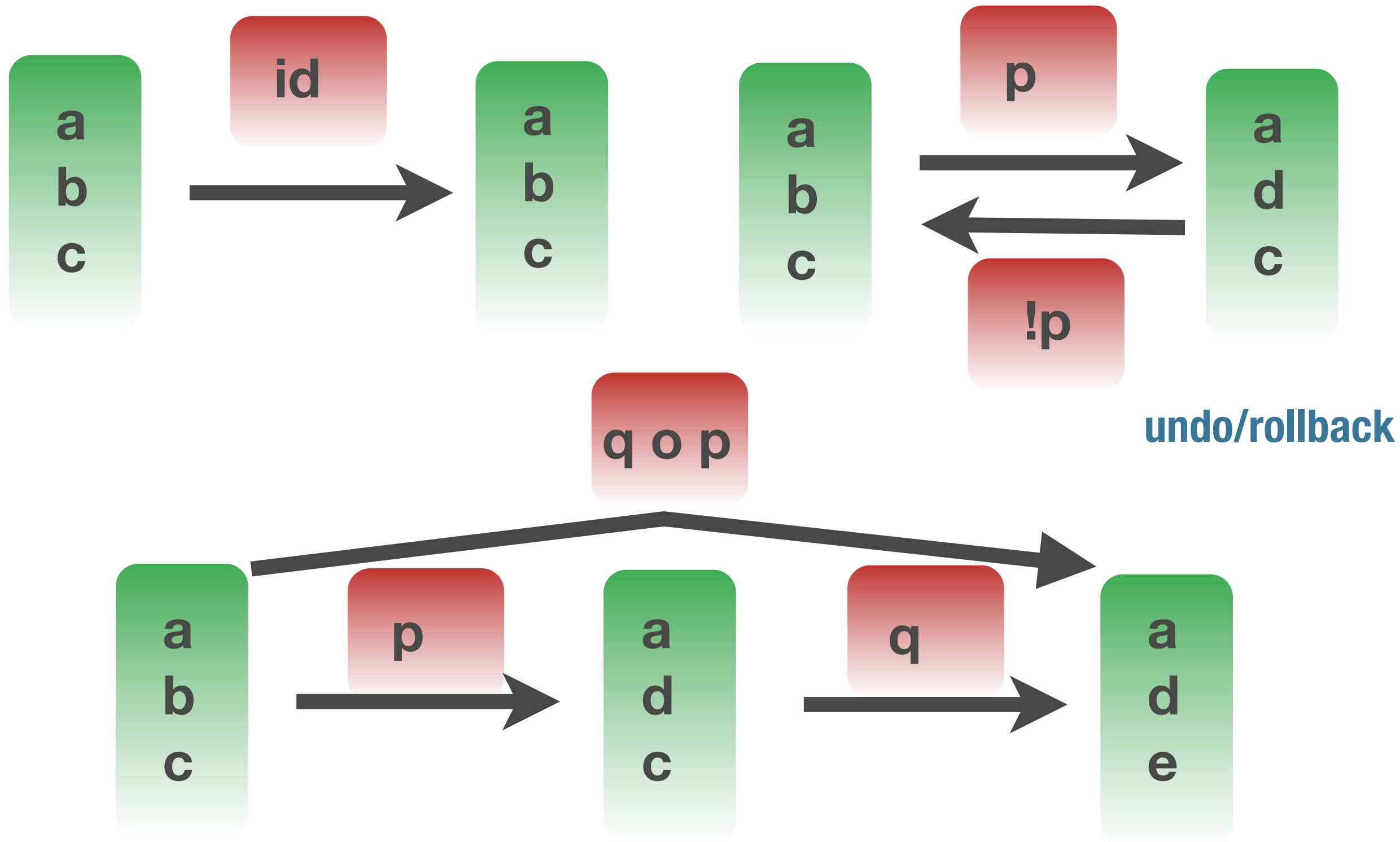






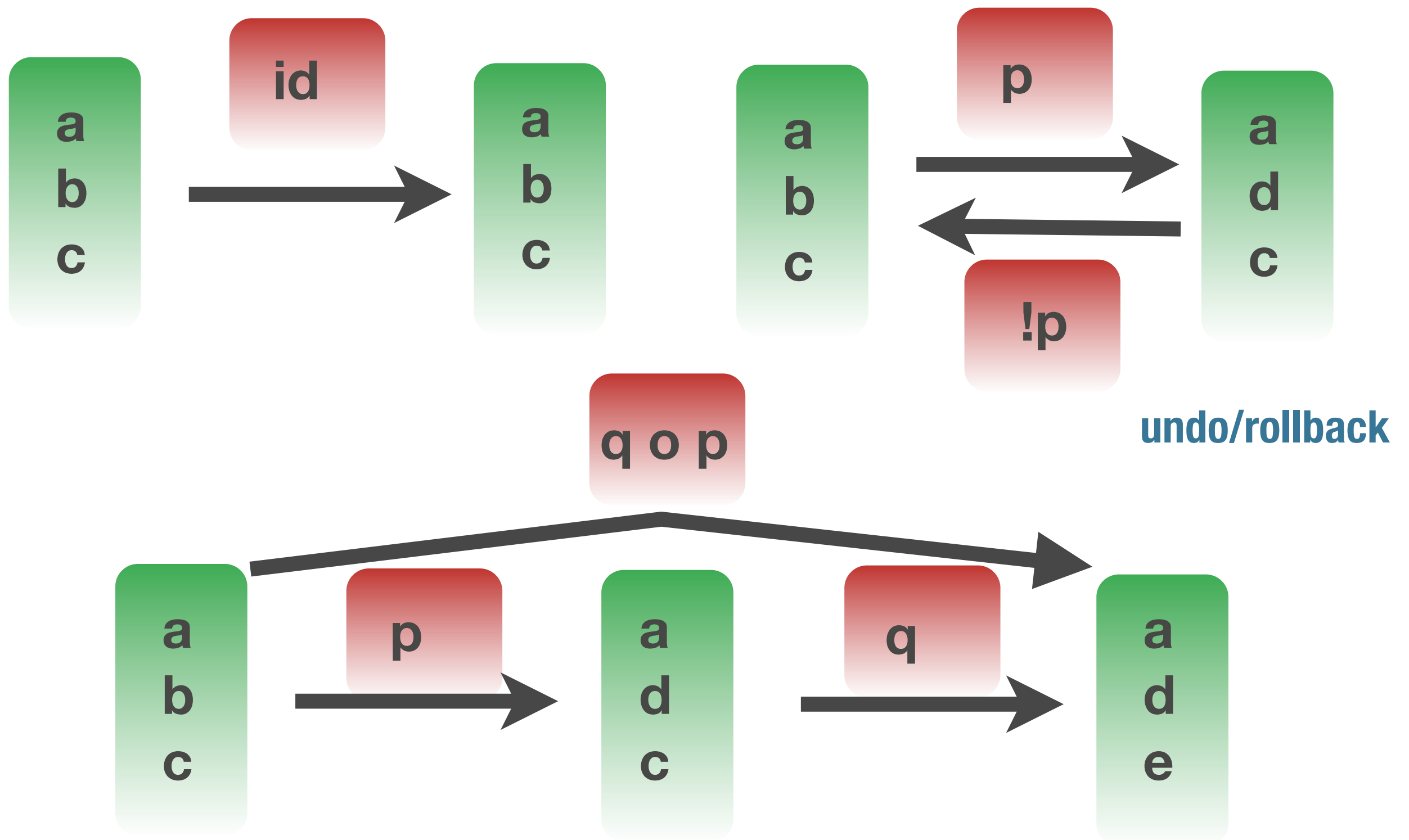




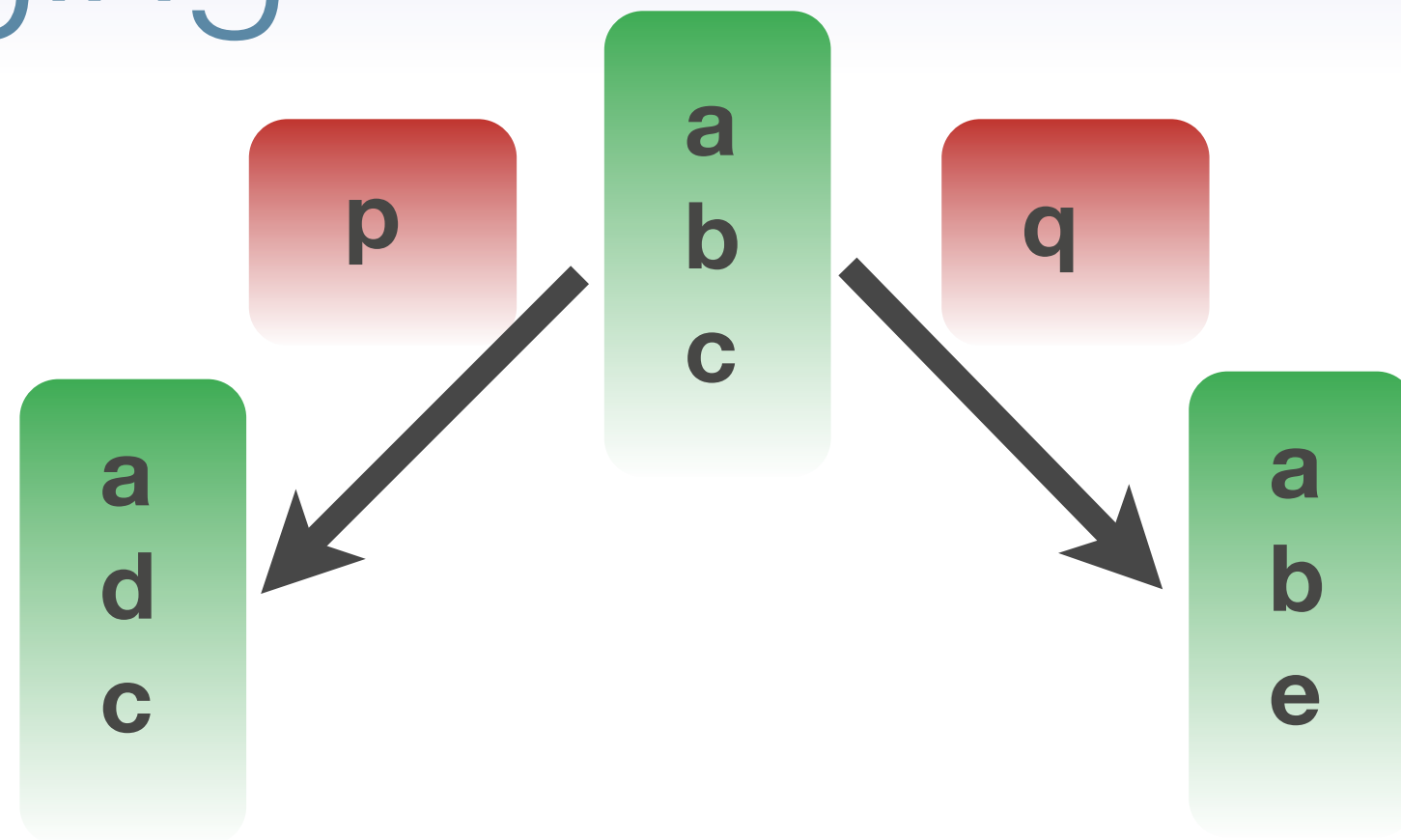




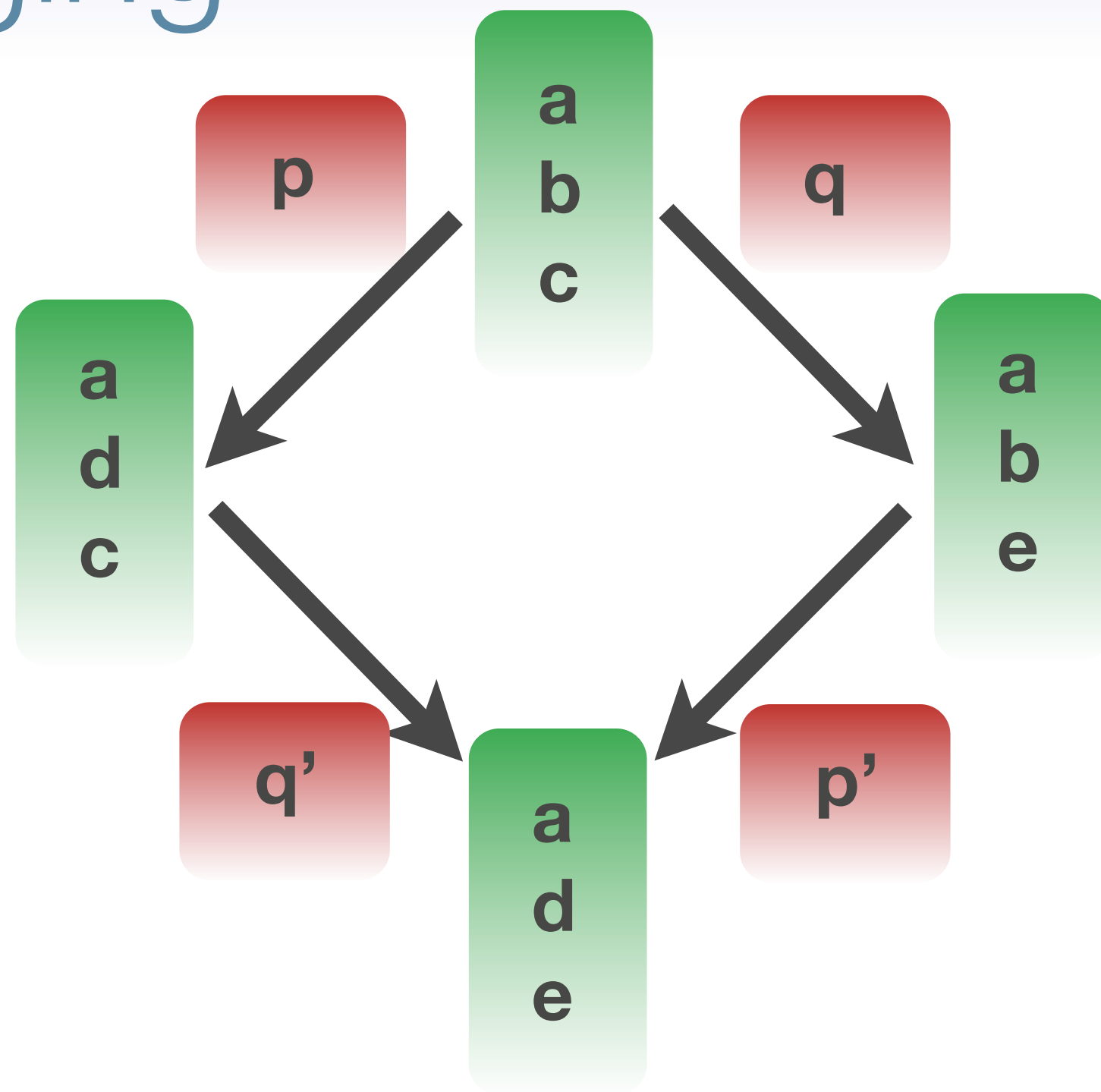
# Patches are paths



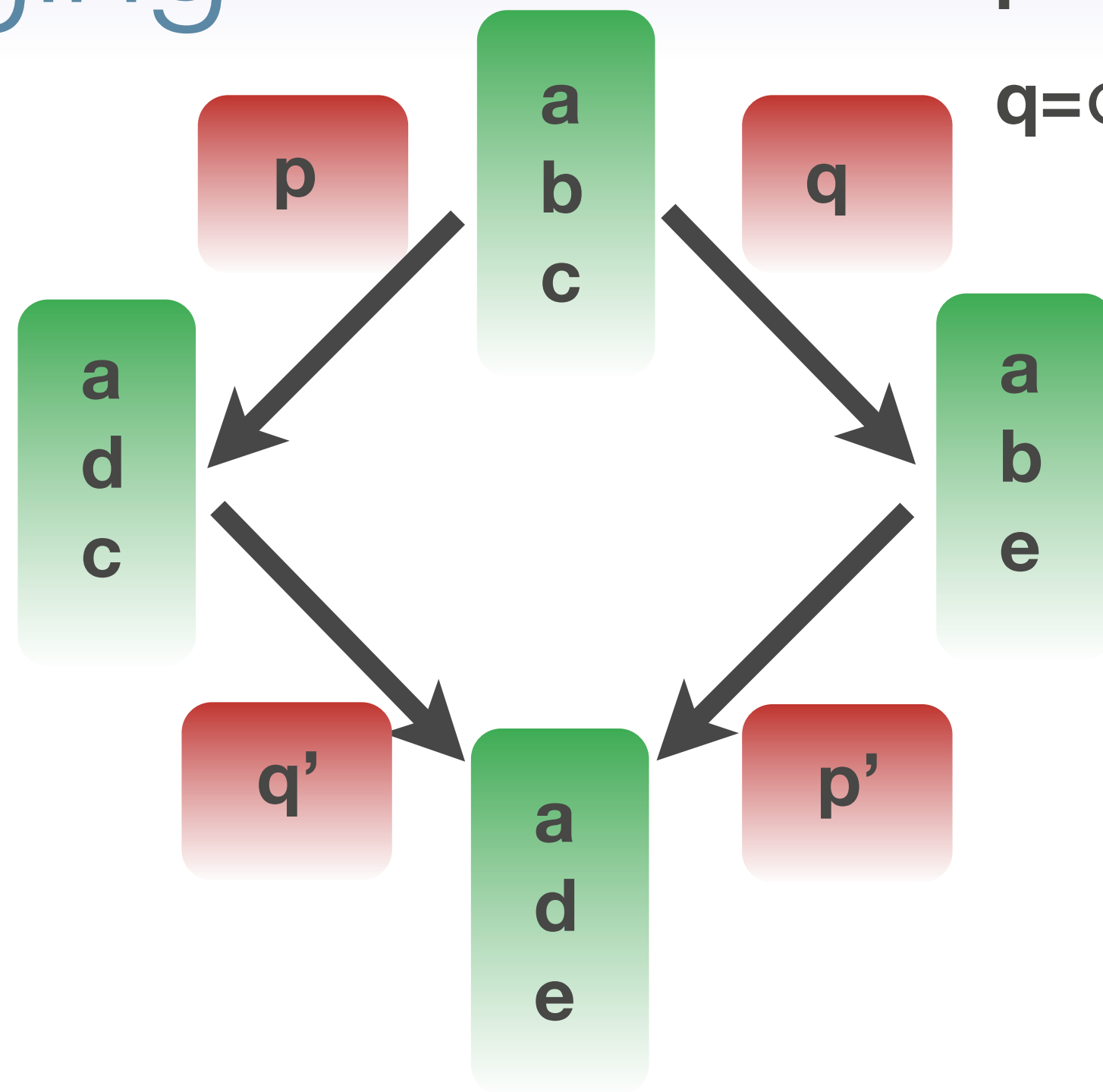
# Merging



# Merging



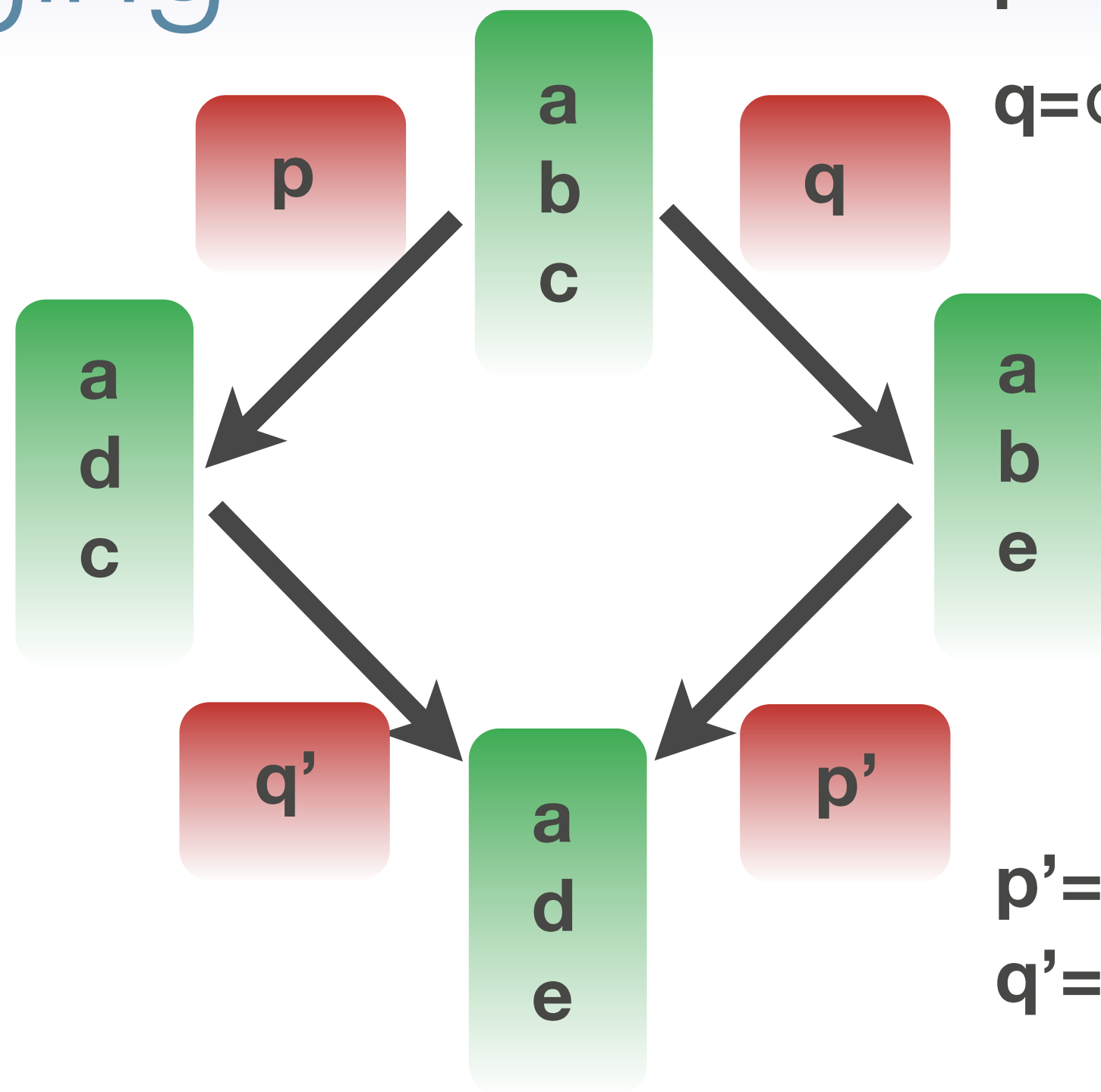
# Merging



$p=b \leftrightarrow d$  at 1

$q=c \leftrightarrow e$  at 2

# Merging



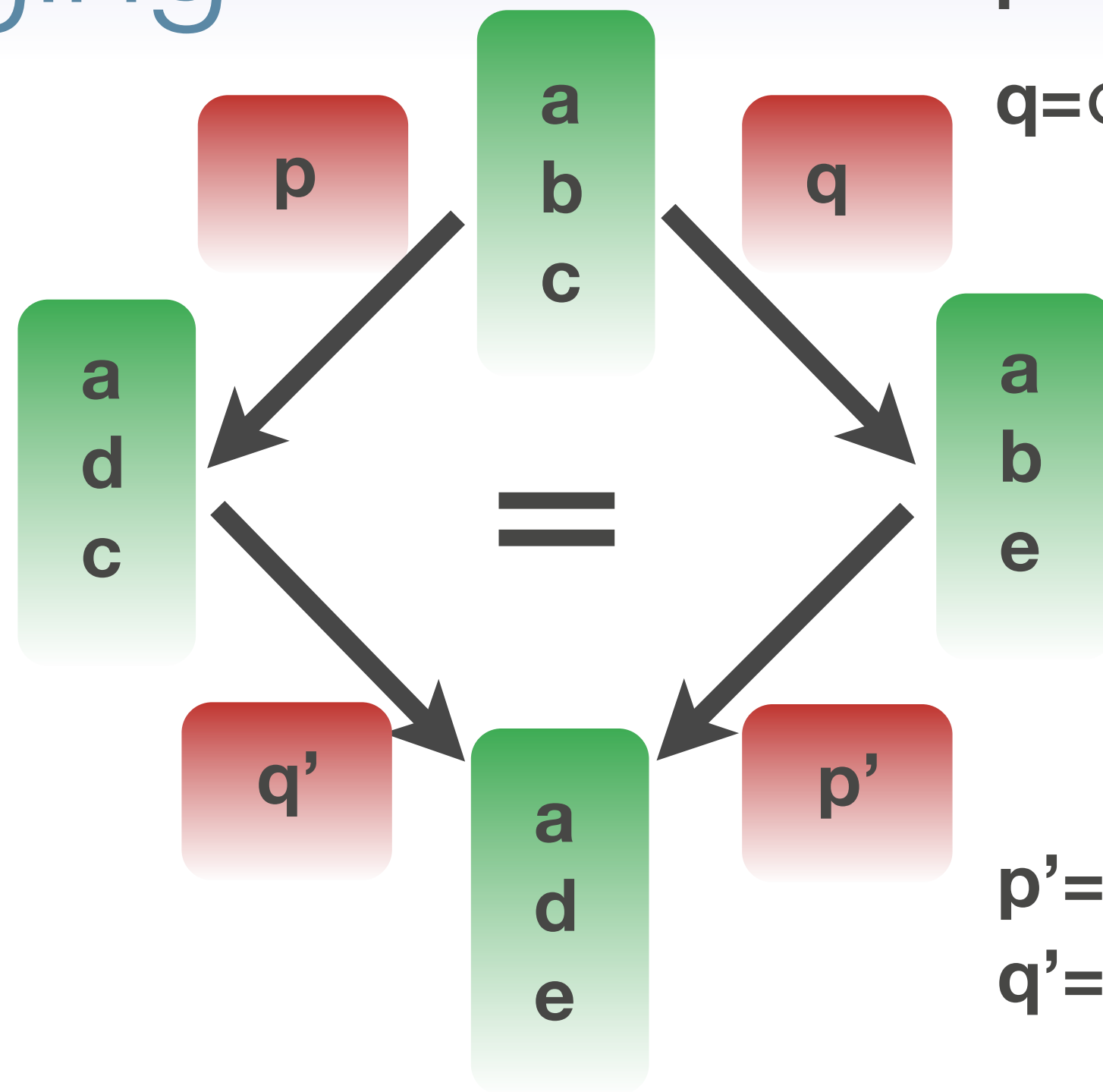
$p=b \leftrightarrow d$  at 1

$q=c \leftrightarrow e$  at 2

$p'=p$

$q'=q$

# Merging



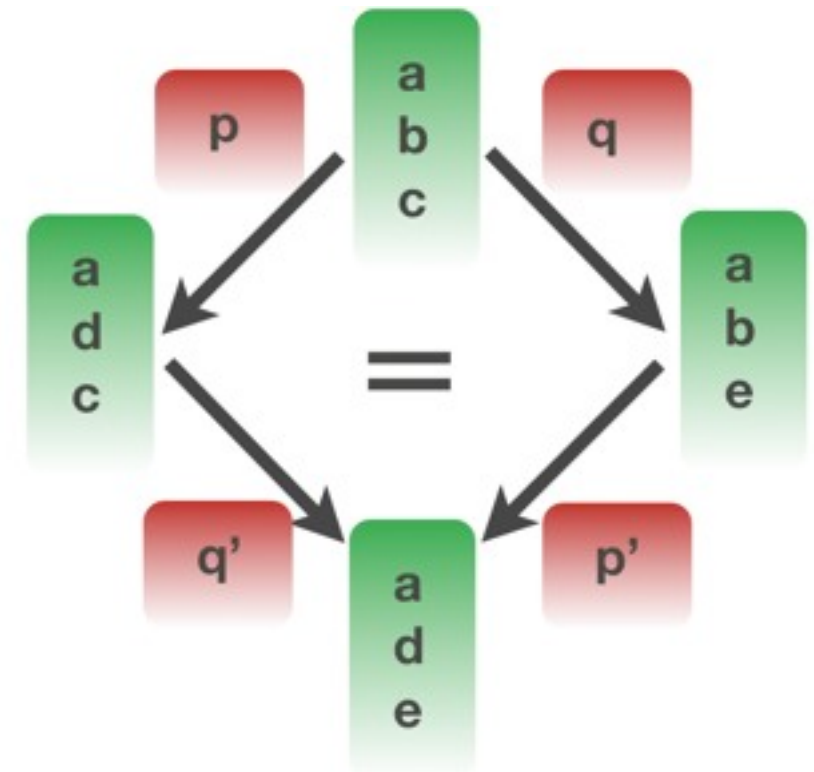
$p=b \leftrightarrow d$  at 1

$q=c \leftrightarrow e$  at 2

$p'=p$   
 $q'=q$

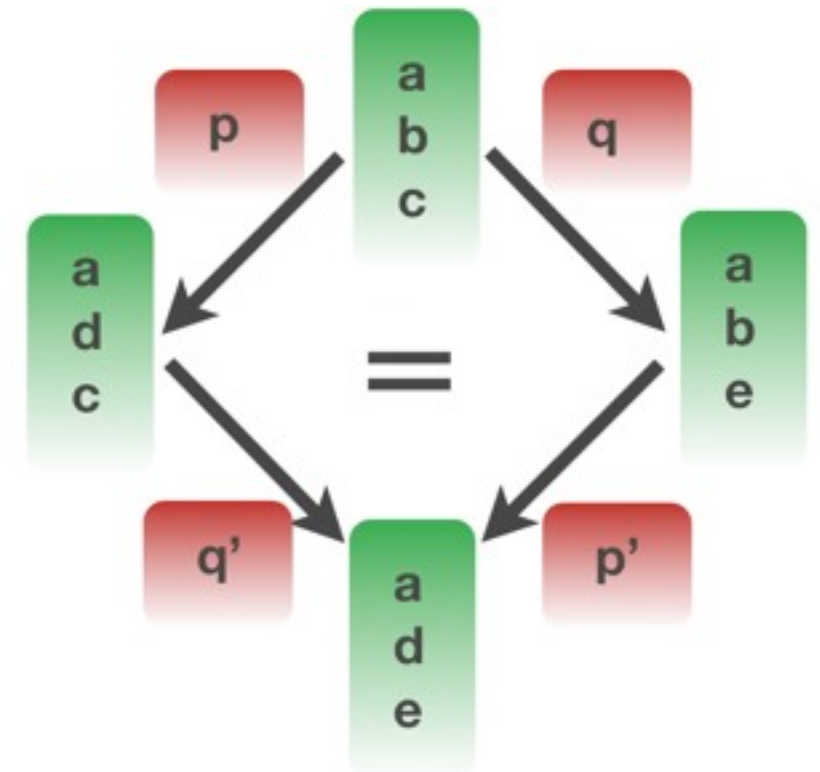
# Merging

merge : (p q : Patch)  
→  $\Sigma q', p' : \text{Patch}.$   
Maybe( $q' \circ p =$   
           $p' \circ q$ )



# Merging

merge : (p q : Patch)  
→  $\Sigma q', p' : \text{Patch}.$   
Maybe( $q' \circ p$  =  
p'  $\circ q$ )



***Equational theory of patches  
= paths between paths***



# Basic Patches

f	i	b	r	a	t	i	o	n
---	---	---	---	---	---	---	---	---

$a \leftrightarrow b @ 2$

f	i	b
---	---	---

f	i	a
---	---	---

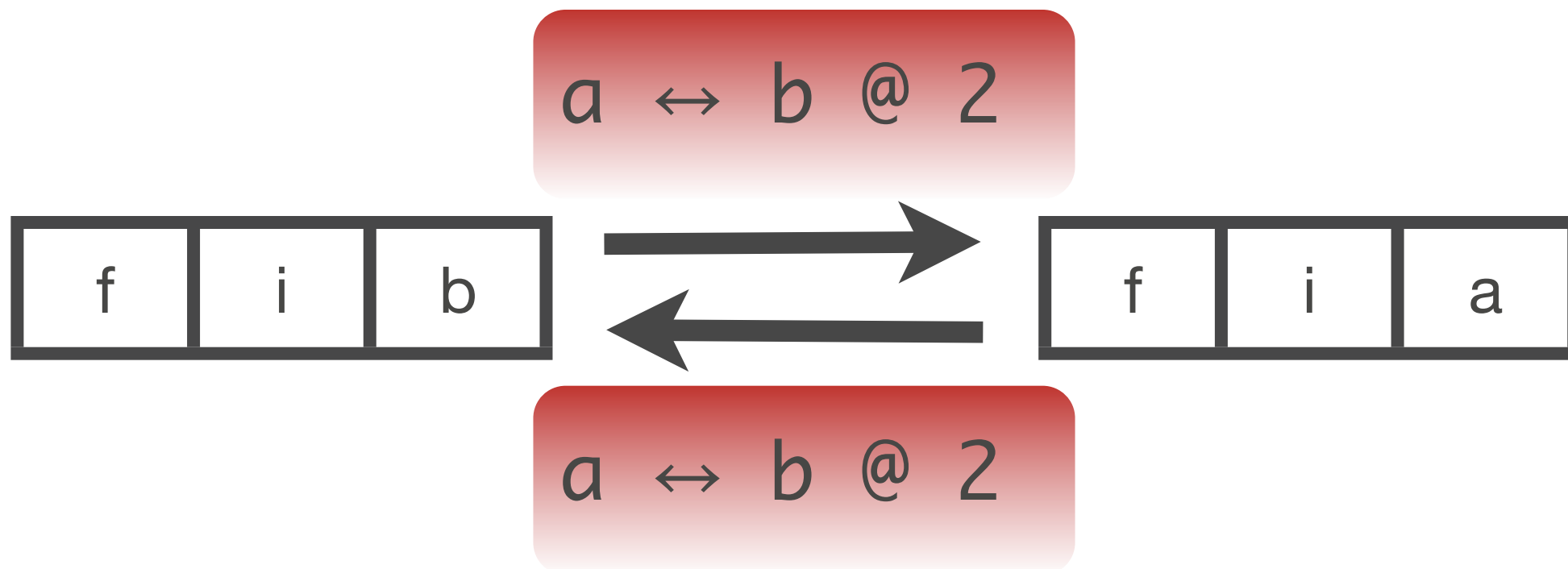
$a \leftrightarrow b @ 2$

# Basic Patches

- \* “Repository” is a char vector of length n

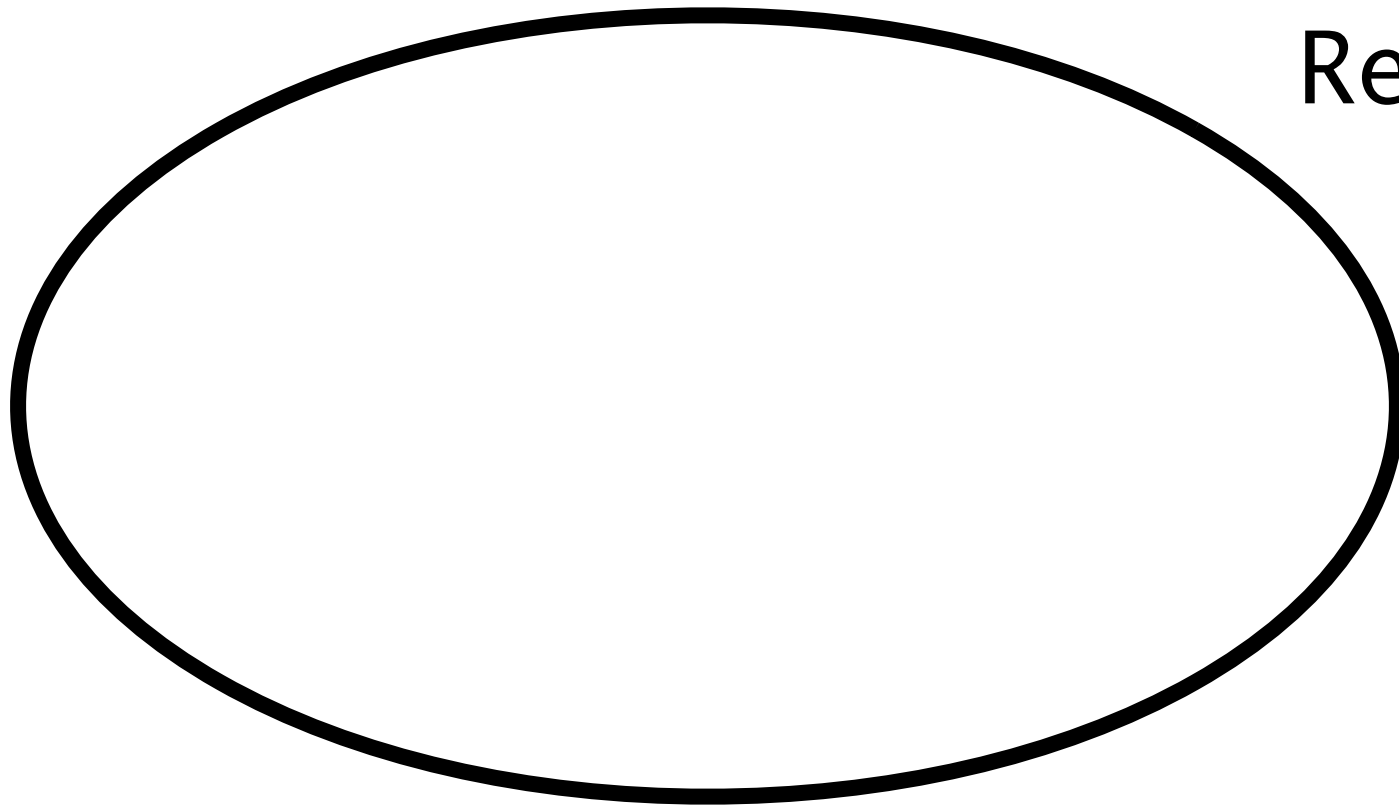


- \* Basic patch is  $a \leftrightarrow b @ i$  where  $i < n$



# Patches as a HIT

Repos : Type



# Patches as a HIT

Repos : Type

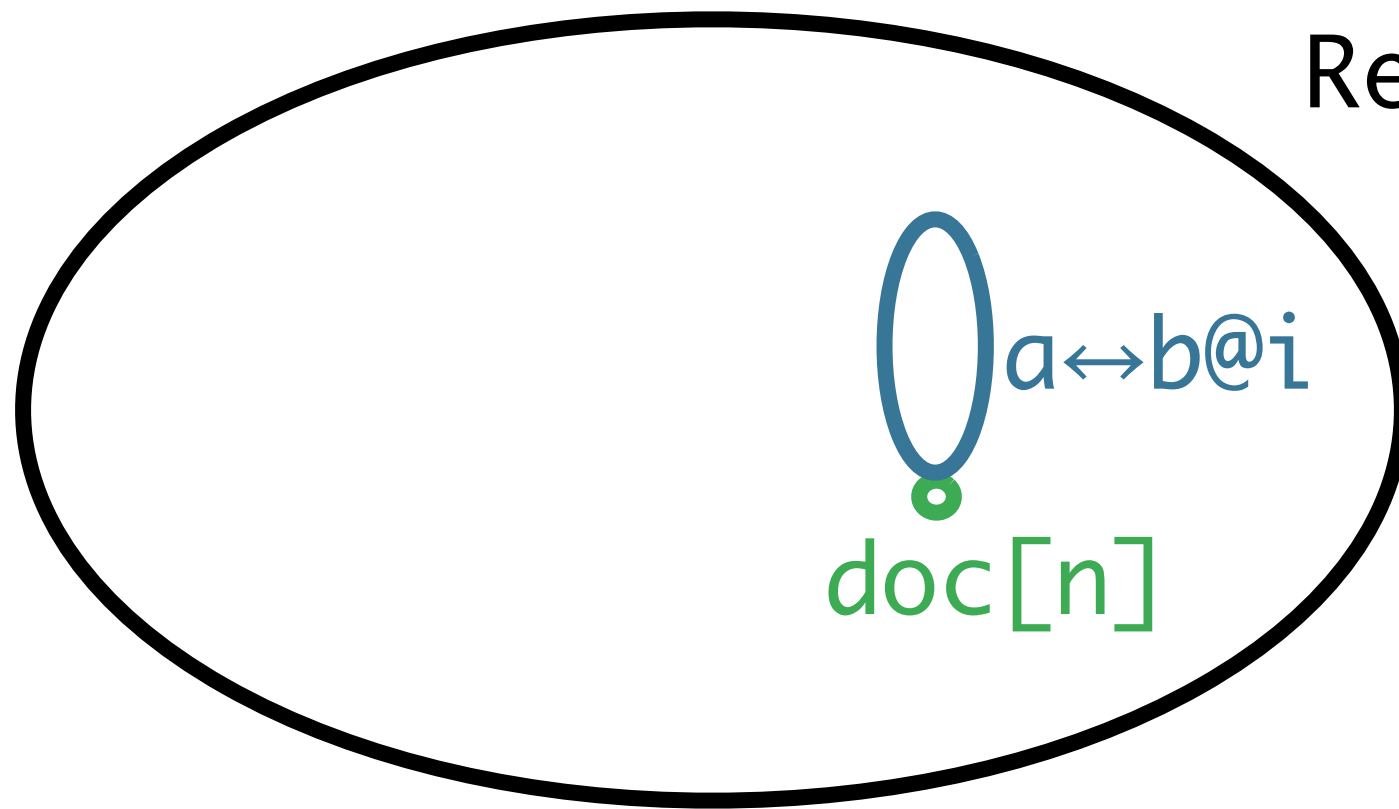


doc[n]

points describe  
repository contents

# Patches as a HIT

Repos : Type

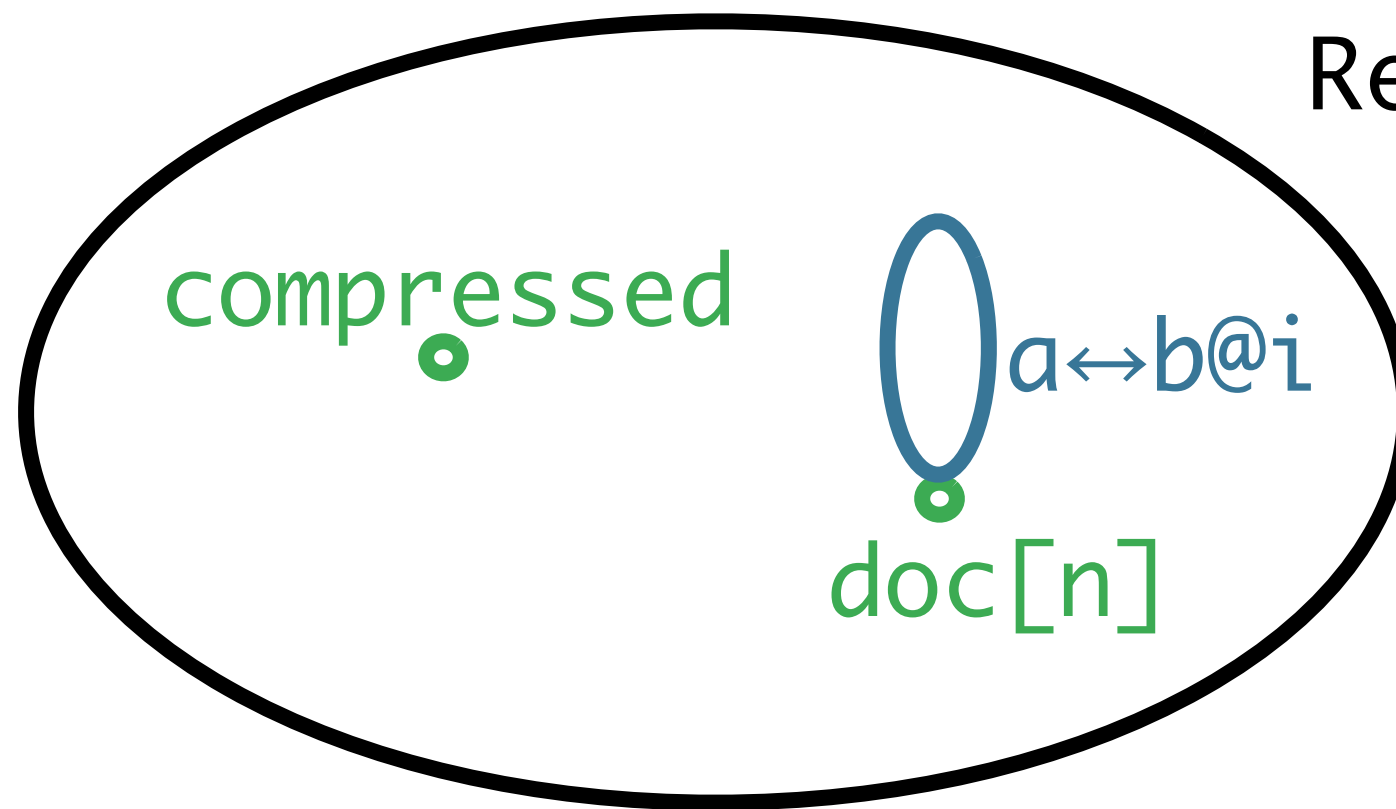


points describe  
repository contents

paths are patches

# Patches as a HIT

Repos : Type

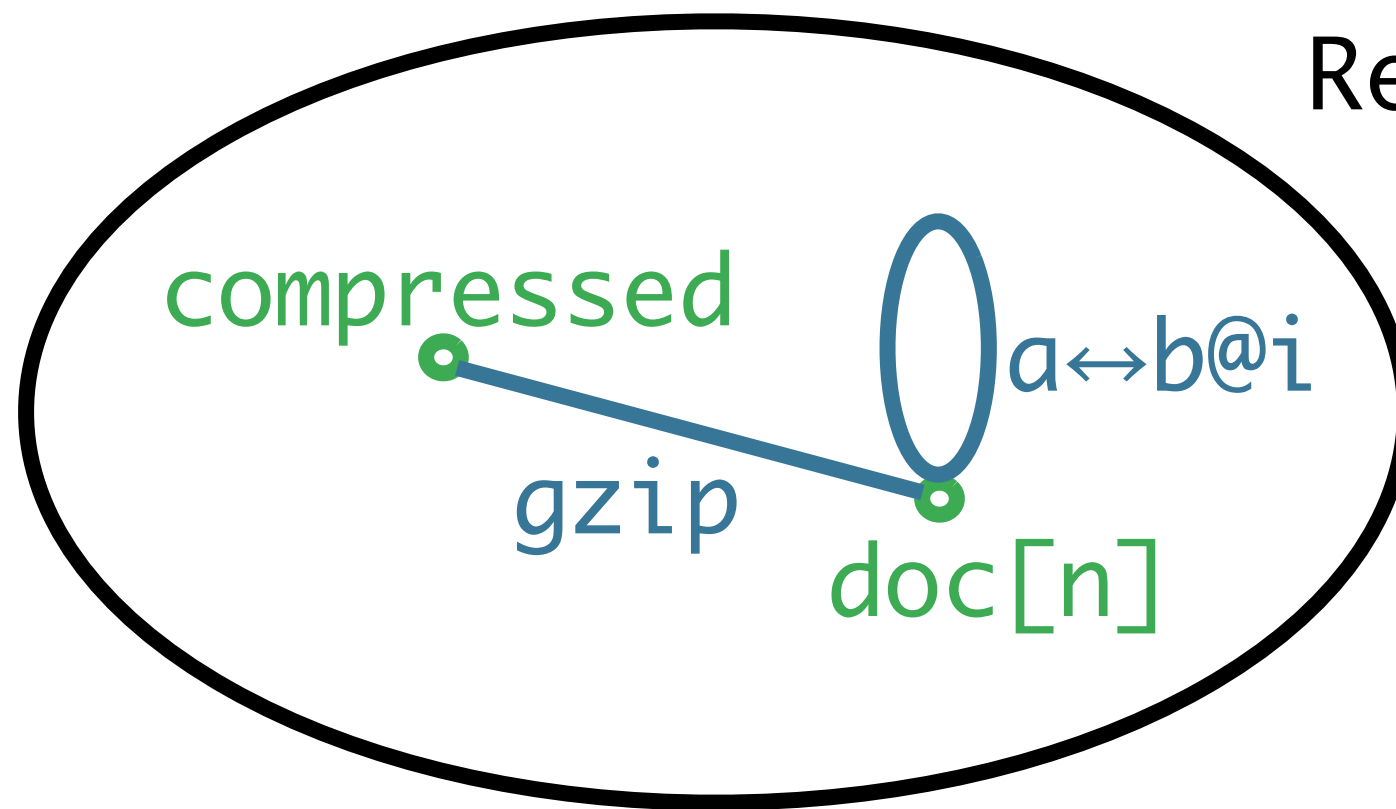


points describe  
repository contents

paths are patches

# Patches as a HIT

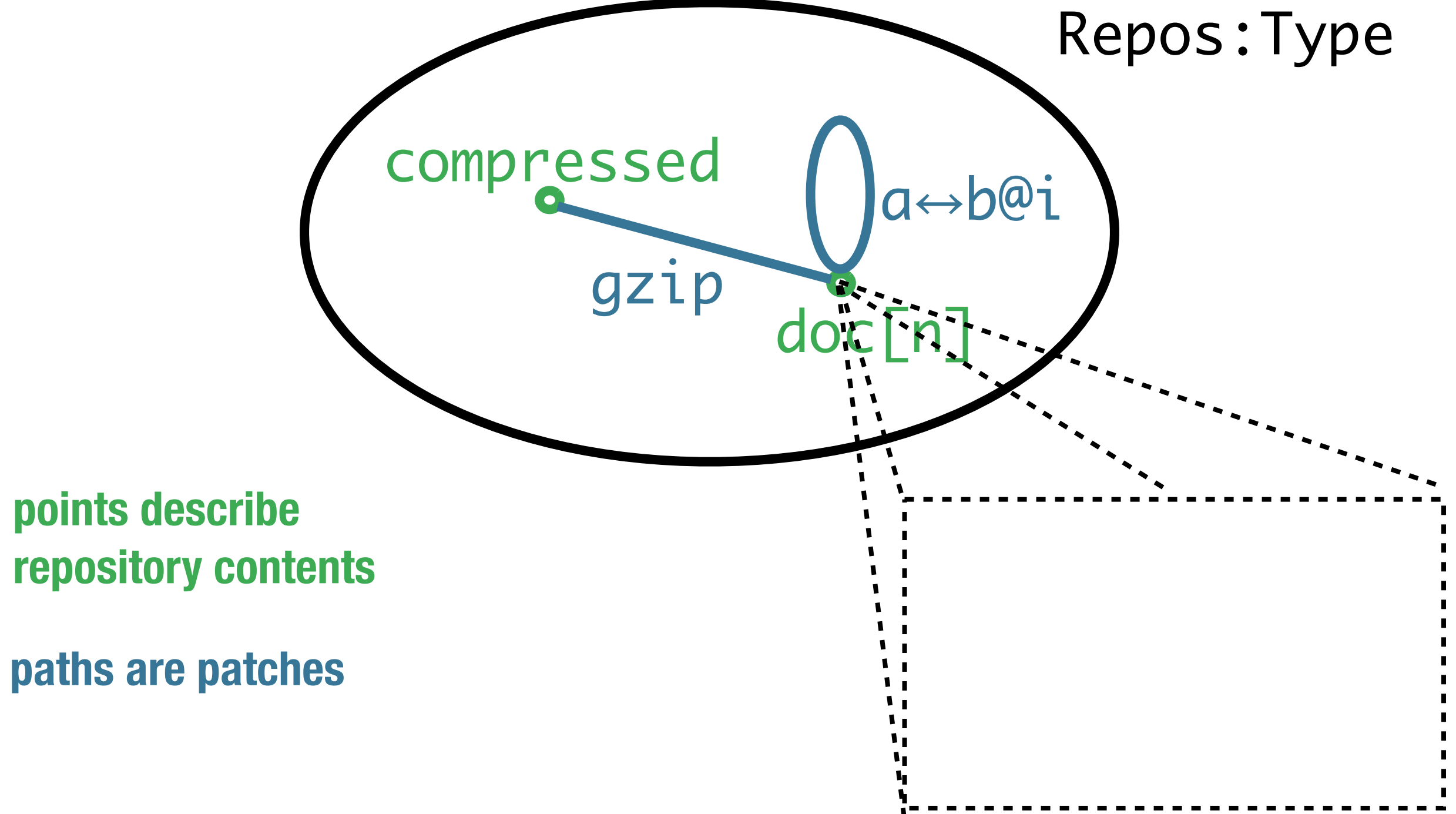
Repos : Type



points describe  
repository contents

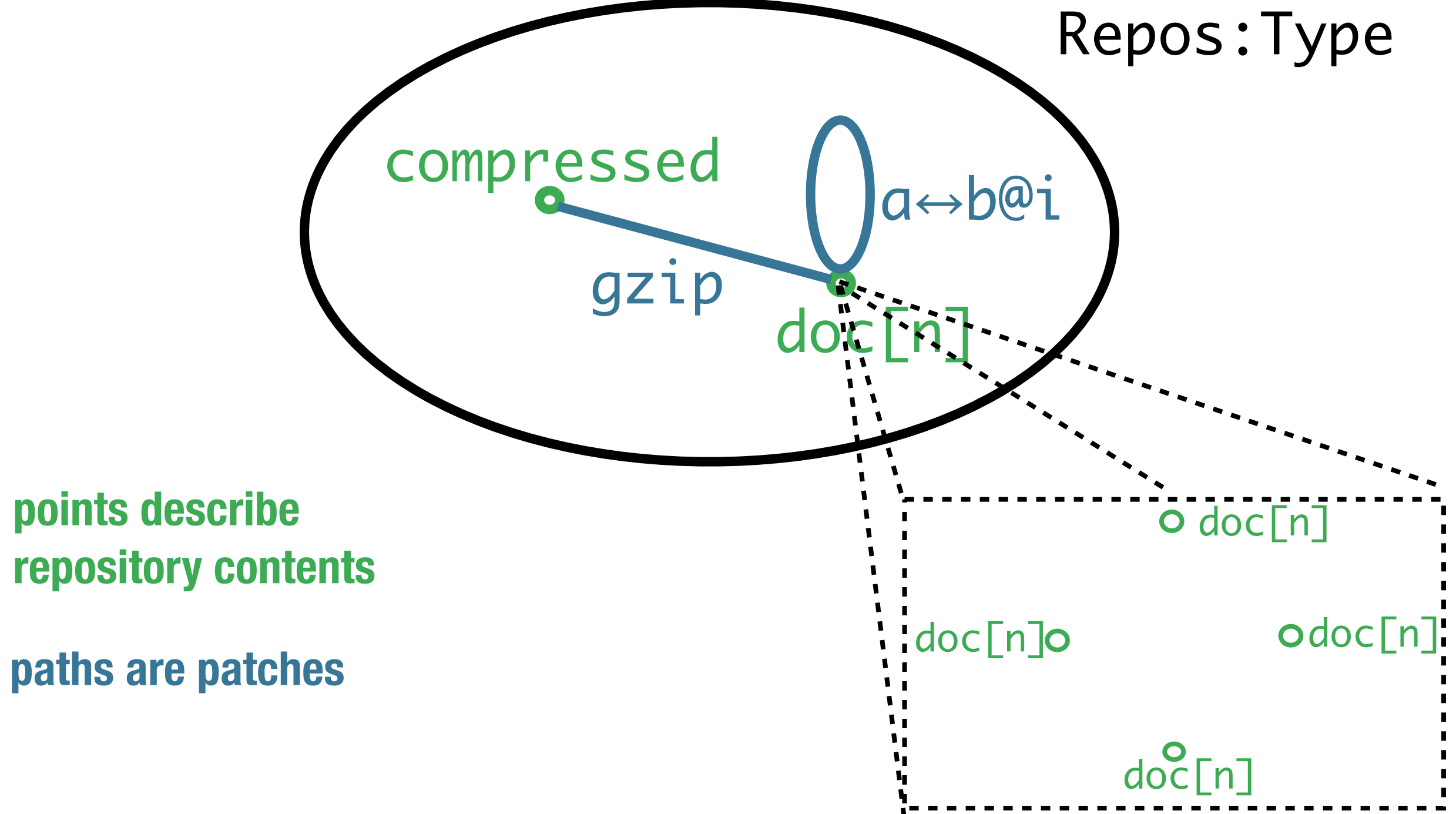
paths are patches

# Patches as a HIT

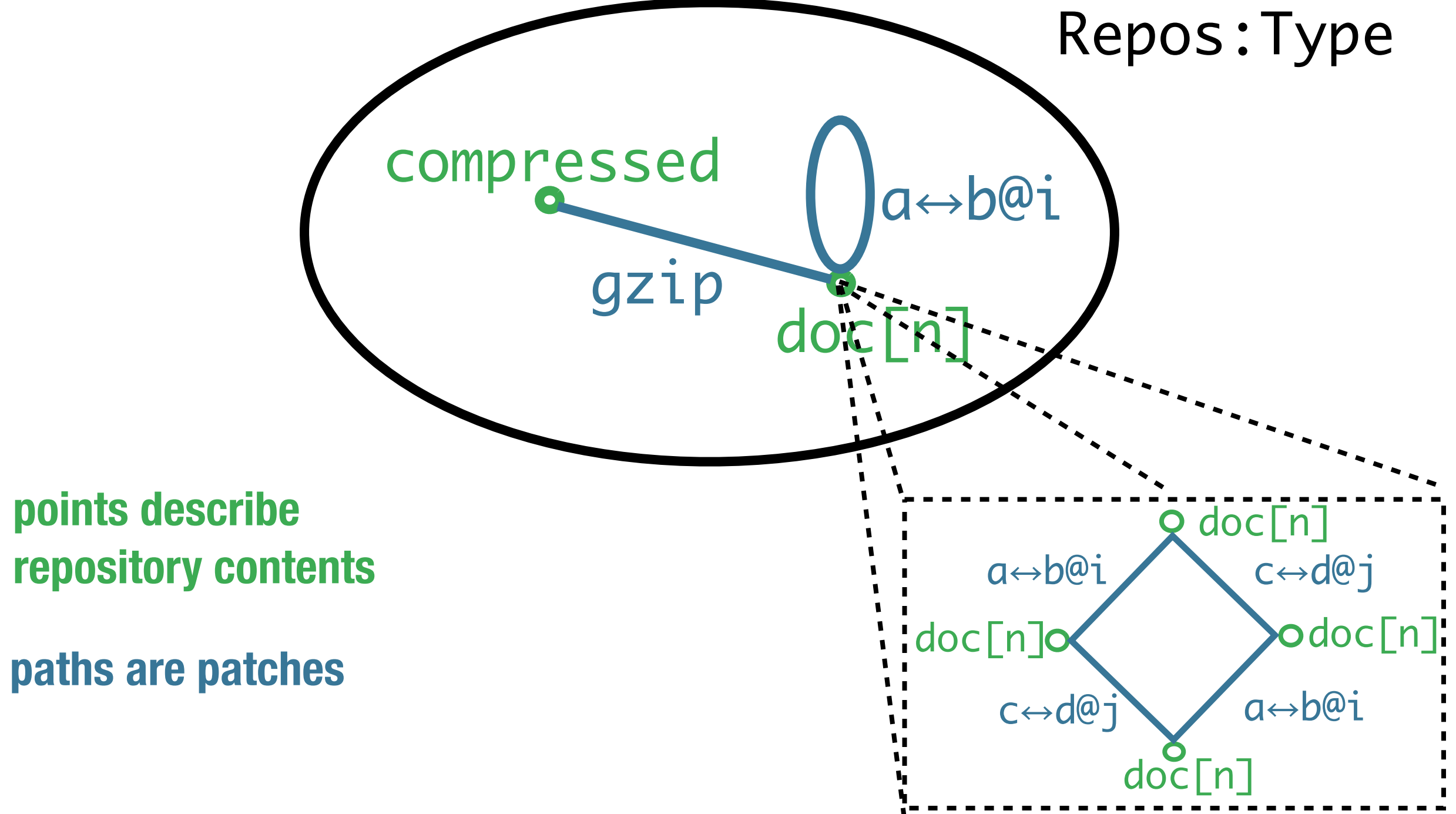




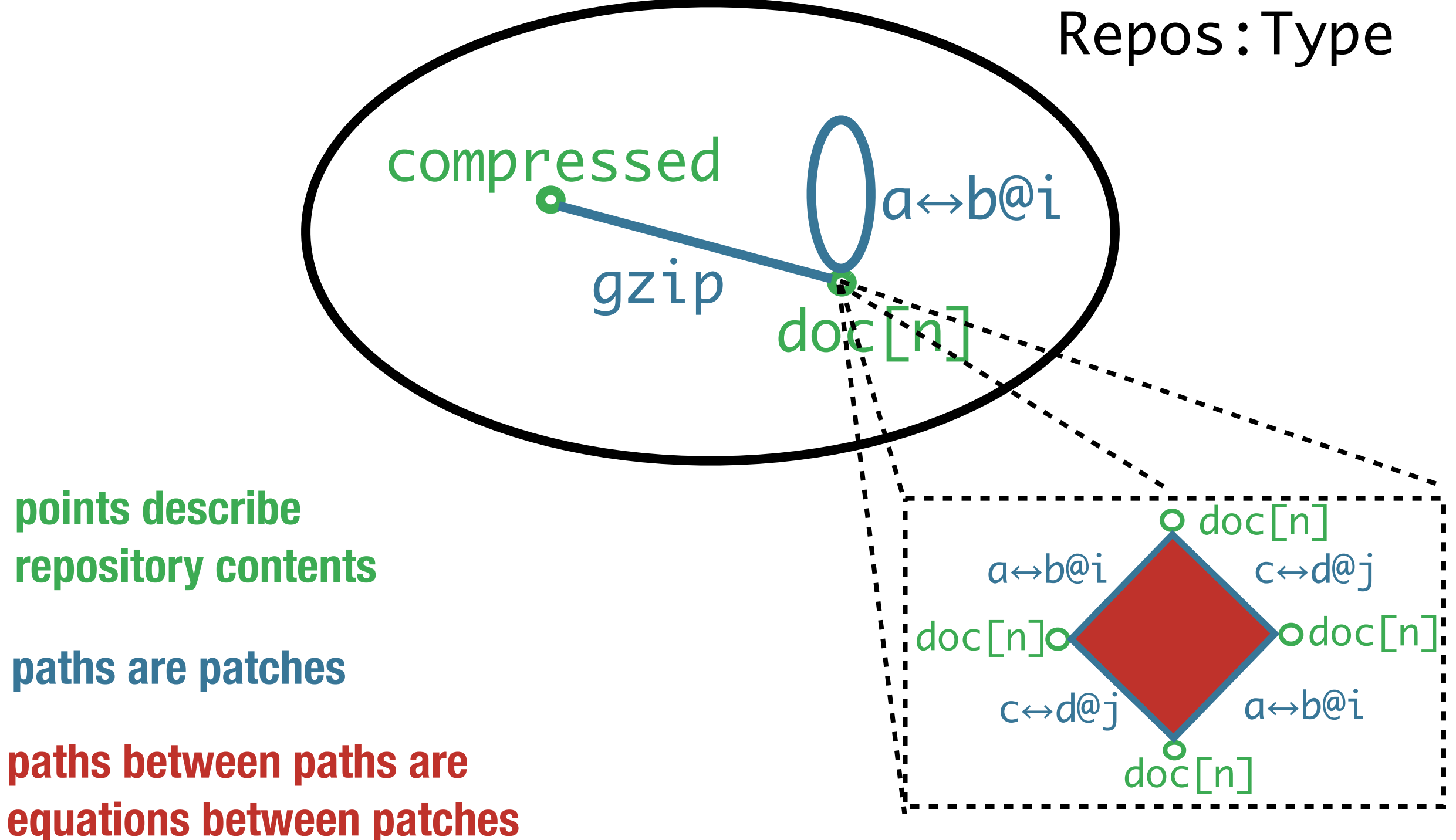
# Patches as a HIT



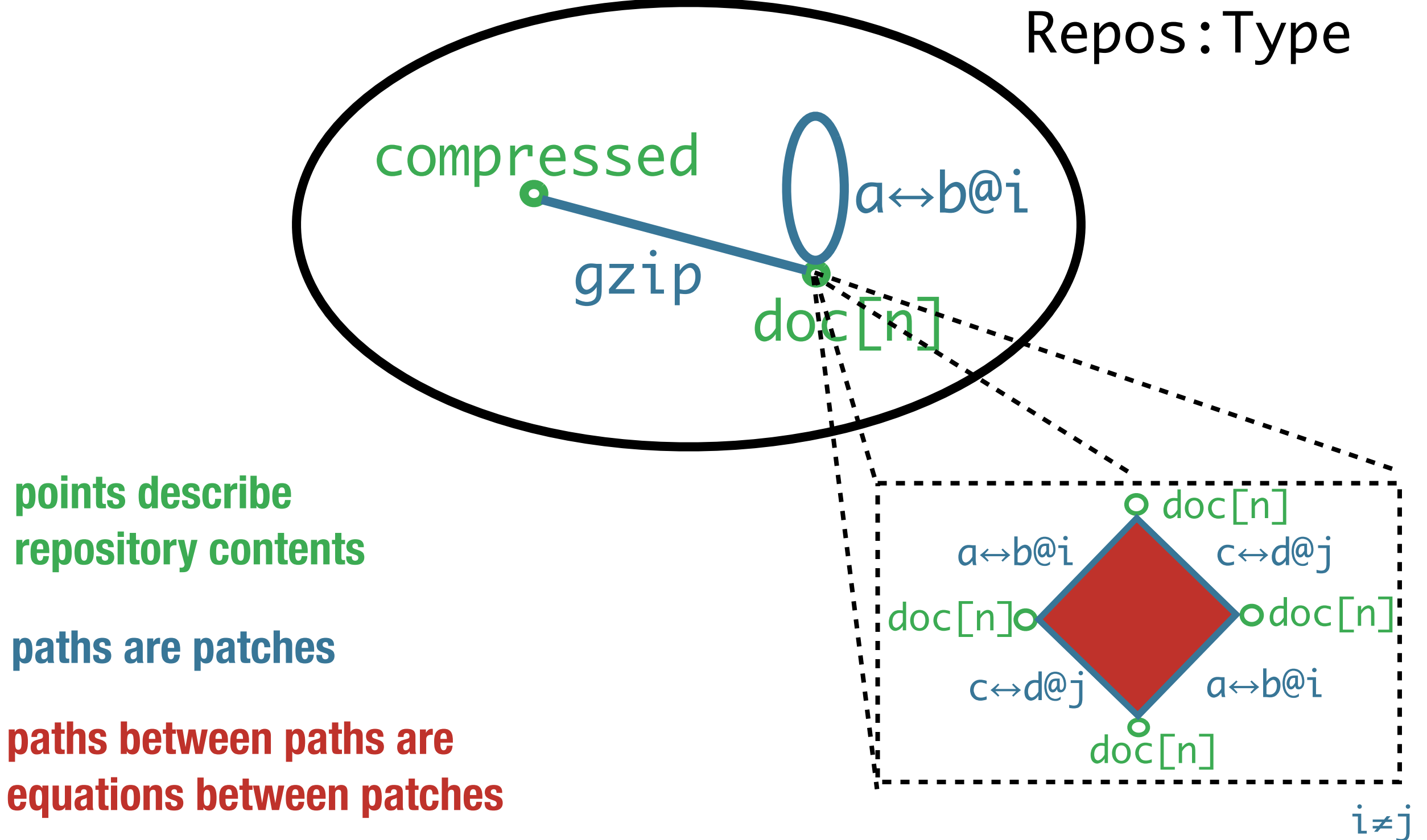
# Patches as a HIT



# Patches as a HIT



# Patches as a HIT



# Generators for HIT

# Generators for HIT

Repos : Type

# Generators for HIT

Repos : Type

doc[n] : Repos

compressed : Repos

# Generators for HIT

Repos : Type

`doc[n]` : Repos

`compressed` : Repos

`(a↔b@i)` : `doc[n] = doc[n]` if `a,b:Char`, `i<n`

`gzip` : `doc[n] = compressed`



# Generators for HIT

Repos : Type

$\text{doc}[n]$  : Repos

$\text{compressed}$  : Repos

$(a \leftrightarrow b @ i)$  :  $\text{doc}[n] = \text{doc}[n]$  if  $a, b : \text{Char}, i < n$

$\text{gzip}$  :  $\text{doc}[n] = \text{compressed}$

**commute:**

$(a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j)$  if  $i \neq j$   
 $= (c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } i)$

## Type: Patch

### Elements:

```
id      : Patch
_°_     : Patch → Patch → Patch
!       : Patch → Patch
_↔_at_  : Char → Char → Fin n → Patch
```

### Equality:

$$(a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j) = \\ (c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } i)$$

```
...
id o p = p = p o id
po(qor) = (poq)or
!p o p = id = p o !p
p=p
p=q if q=p
p=r if p=q and q=r
!p = !p' if p = p'
p o q = p' o q' if p = p' and q = q'
```

## Type: Repos

Points:  $\text{doc}[n]$

### Paths:

$$a \leftrightarrow b @ i$$

### Paths between paths:

commute :

$$(a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j) = \\ (c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } i)$$

# Repos recursion

To define a function  $\text{Repos} \rightarrow A$   
it suffices to

# Repos recursion

To define a function  $\text{Repos} \rightarrow A$   
it suffices to

- \* map the element generators of  $\text{Repos}$   
to elements of  $A$

# Repos recursion

To define a function  $\text{Repos} \rightarrow A$   
it suffices to

- \* map the element generators of  $\text{Repos}$   
to elements of  $A$
- \* map the equality generators of  $\text{Repos}$   
to equalities between the corresponding elements of  $A$

# Repos recursion

To define a function  $\text{Repos} \rightarrow A$   
it suffices to

- \* map the element generators of  $\text{Repos}$  to elements of  $A$
- \* map the equality generators of  $\text{Repos}$  to equalities between the corresponding elements of  $A$
- \* map the equality-between-equality generators to equalities between the corresponding equalities in  $A$

# Repos recursion

To define a function  $\text{Repos} \rightarrow A$   
it suffices to

- \* map the element generators of  $\text{Repos}$  to elements of  $A$
- \* map the equality generators of  $\text{Repos}$  to equalities between the corresponding elements of  $A$
- \* map the equality-between-equality generators to equalities between the corresponding equalities in  $A$

*All functions on  $\text{Repos}$  respect patches*

*All functions on patches respect patch equality*

# Interpreter

Goal is to define:

$\text{interp} : \text{doc}[n] = \text{doc}[n]$   
 $\rightarrow \text{Bijection } (\text{Vec Char } n) (\text{Vec Char } n)$



# Interpreter

Goal is to define:

$\text{interp} : \text{doc}[n] = \text{doc}[n]$

$\rightarrow \text{Bijection } (\text{Vec Char } n) (\text{Vec Char } n)$

$\text{interp}(\text{id}) = (\lambda x.x, \dots)$

$\text{interp}(q \circ p) = (\text{interp } q) \circ_b (\text{interp } p)$

$\text{interp}(!p) = !_b (\text{interp } p)$

$\text{interp}(a \leftrightarrow b @ i) = \text{swapat } a \ b \ i$

# Interpreter

Goal is to define:

$\text{interp} : \text{doc}[n] = \text{doc}[n]$   
     $\rightarrow \text{Bijection } (\text{Vec Char } n) (\text{Vec Char } n)$   
 $\text{interp}(\text{id}) = (\lambda x.x, \dots)$   
 $\text{interp}(q \circ p) = (\text{interp } q) \circ_b (\text{interp } p)$   
 $\text{interp}(!p) = !_b (\text{interp } p)$   
 $\text{interp}(a \leftrightarrow b @ i) = \text{swapat } a \ b \ i$

*But only tool available is RepoDesc recursion:  
no direct recursion over paths*

$\text{interp} : \text{doc}[n] = \text{doc}[n]$   
 $\rightarrow \text{Bijection } (\text{Vec Char } n) (\text{Vec Char } n)$   
 $\text{interp}(a \leftrightarrow b \text{ at } i) = \text{swapat } a \ b \ i$

*Need to pick A and define*

$I(\text{doc}[n]) := \dots : A$

$I_1(a \leftrightarrow b @ i) := \dots : I(\text{doc}[n]) = I(\text{doc}[n])$

$I_2(\text{compose}) := \dots$

$\text{interp} : \text{doc}[n] = \text{doc}[n]$   
 $\rightarrow \text{Bijection } (\text{Vec Char } n) (\text{Vec Char } n)$   
 $\text{interp}(a \leftrightarrow b \text{ at } i) = \text{swapat } a \ b \ i$

*Key idea: pick  $A = \text{Type}$  and define*

$I(\text{doc}[n]) := \dots : \text{Type}$

$I_1(a \leftrightarrow b @ i) := \dots : I(\text{doc}[n]) = I(\text{doc}[n])$

$I_2(\text{compose}) := \dots$

interp : doc[n]=doc[n]  
→ Bijection (Vec Char n) (Vec Char n)  
interp(a↔b at i) = swapat a b i

*Key idea: pick A = Type and define*

I(doc[n]) := Vec Char n : Type

I<sub>1</sub>(a↔b@i) := ... : I(doc[n]) = I(doc[n])

I<sub>2</sub>(compose) := ...

$\text{interp} : \text{doc}[n] = \text{doc}[n]$   
 $\rightarrow \text{Bijection } (\text{Vec Char } n) (\text{Vec Char } n)$   
 $\text{interp}(a \leftrightarrow b \text{ at } i) = \text{swapat } a \ b \ i$

*Key idea: pick  $A = \text{Type}$  and define*

$I(\text{doc}[n]) := \text{Vec Char } n : \text{Type}$

$I_1(a \leftrightarrow b @ i) := \dots : \text{Vec Char } n = \text{Vec Char } n$

$I_2(\text{compose}) := \dots$

$\text{interp} : \text{doc}[n] = \text{doc}[n]$   
 $\rightarrow \text{Bijection } (\text{Vec Char } n) (\text{Vec Char } n)$   
 $\text{interp}(a \leftrightarrow b \text{ at } i) = \text{swapat } a \ b \ i$

*Key idea: pick  $A = \text{Type}$  and define*

$I(\text{doc}[n]) := \text{Vec Char } n : \text{Type}$

$I_1(a \leftrightarrow b @ i) := \text{ua}(\text{swapat } a \ b \ i)$

$: \text{Vec Char } n = \text{Vec Char } n$

$I_2(\text{compose}) := \dots$

`interp : doc[n]=doc[n]`  
     $\rightarrow$  `Bijection (Vec Char n) (Vec Char n)`  
`interp(a $\leftrightarrow$ b at i) = swapat a b i`

*Key idea: pick  $A = \text{Type}$  and define*

`I(doc[n]) := Vec Char n : Type`

`I1(a $\leftrightarrow$ b@i) := ua(swapat a b i)`

`: Vec Char n = Vec Char n`

`I2(compose) := ...`

**univalence**





$\text{interp} : \text{doc}[n] = \text{doc}[n]$   
 $\rightarrow \text{Bijection } (\text{Vec Char } n) (\text{Vec Char } n)$   
 $\text{interp}(a \leftrightarrow b \text{ at } i) = \text{swapat } a \ b \ i$

*Key idea: pick  $A = \text{Type}$  and define*

$I(\text{doc}[n]) := \text{Vec Char } n : \text{Type}$

$I_1(a \leftrightarrow b @ i) := \text{ua}(\text{swapat } a \ b \ i)$

$: \text{Vec Char } n = \text{Vec Char } n$

$I_2(\text{compose}) := \text{<proof about swapat>}$

$\text{interp} : \text{doc}[n] = \text{doc}[n]$   
 $\rightarrow \text{Bijection } (\text{Vec Char } n) (\text{Vec Char } n)$

$\text{interp}(p) = \text{ua}^{-1}(\text{I}_1(p))$

*Key idea: pick  $A = \text{Type}$  and define*

$\text{I}(\text{doc}[n]) := \text{Vec Char } n : \text{Type}$

$\text{I}_1(a \leftrightarrow b @ i) := \text{ua}(\text{swapat } a \ b \ i)$

$: \text{Vec Char } n = \text{Vec Char } n$

$\text{I}_2(\text{compose}) := \langle \text{proof about swapat} \rangle$

$\text{interp} : \text{doc}[n] = \text{doc}[n]$   
 $\rightarrow \text{Bijection } (\text{Vec Char } n) (\text{Vec Char } n)$   
 $\text{interp}(p) = \text{ua}^{-1}(\text{I}_1(p))$

Satisfies the desired equations (as propositional equalities):

$\text{interp}(\text{id}) = (\lambda x.x, \dots)$   
 $\text{interp}(q \circ p) = (\text{interp } q) \circ_b (\text{interp } p)$   
 $\text{interp}(!p) = !_b (\text{interp } p)$   
 $\text{interp}(a \leftrightarrow b @ i) = \text{swapat } a \ b \ i$

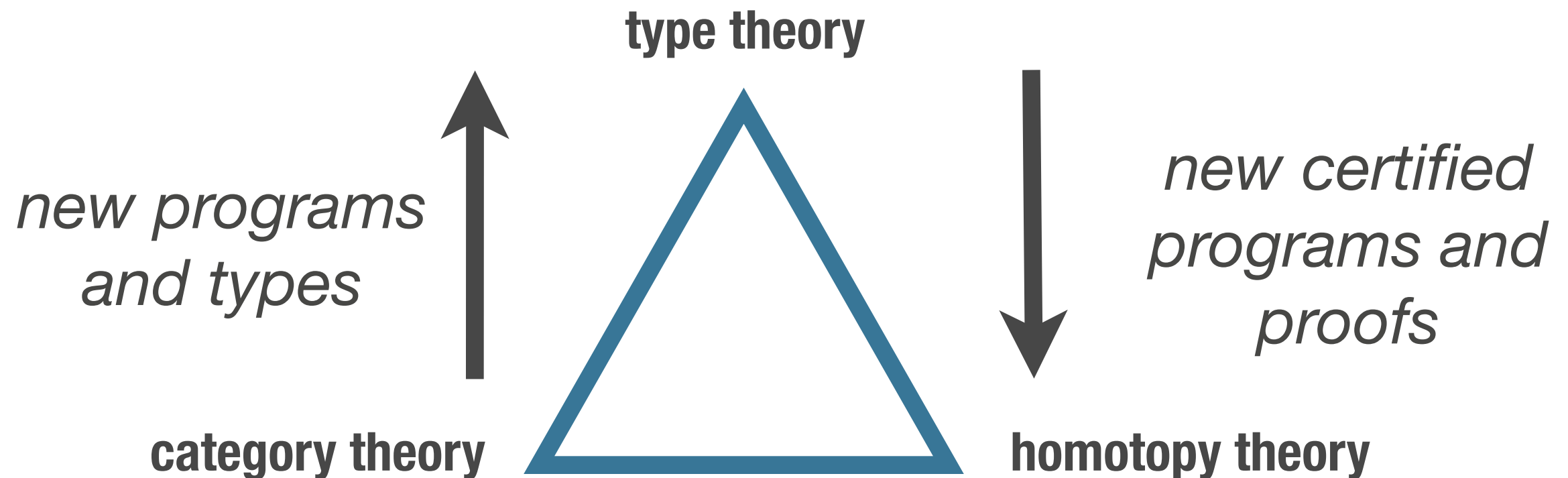
# Summary

- ✱  $I : \text{Repos} \rightarrow \text{Type}$  interprets `Repos` as `Types`, patches as bijections, satisfying patch equalities
- ✱ Higher inductive elim. defines functions that respect equality: you specify what happens on the generators; homomorphically extended to `id`, `o`, `!`, ...
- ✱ Univalence lets you give a computational model of equality proofs (here, patches); guaranteed to satisfy laws
- ✱ Shorter definition and code:  
1 basic patch & 4 basic axioms of equality, instead of  
4 patches & 14 equations

# Operational semantics

- ✱ Can't quite run these programs yet
- ✱ Some special cases known, some recent progress:
  - Licata&Harper, '12
  - Coquand&Barras, '13
  - Shulman, '13
  - Bezem&Coquand&Huber, '13

# Homotopy Type Theory



# Reading list

1.The HoTT Book

2.Homotopy theory in Agda:  
Fundamental group of the circle [LICS'13]

$\pi_n(S^n) = \mathbb{Z}$  [proceedings]  
[github.com/dlicata335/](https://github.com/dlicata335/)

3.Blog: [homotopytypetheory.org](http://homotopytypetheory.org)